IMPROVING BANDWIDTH ALLOCATION IN CLOUD COMPUTING ENVIRONMENTS VIA "BANDWIDTH AS A SERVICE" PARTITIONING SCHEME

by

Anthony Amaro Jr

A thesis submitted to the Graduate College of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
August 2016

Committee Members:

Wuxu Peng, Chair

Xiao Chen

Stan McClellan

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

This work is dedicated to my beautiful wife, Martha Shea Amaro, of whose patience and loving-kindness I am wholly unworthy; my daughters Lillian Elaine and Lana Dee, the lights of my life; my grandparents Aura and Abelardo Castillo, whose bravery and sacrifice allowed our family a chance at a better life; my uncle Marlon Castillo and aunt Lisa Castillo who where there for me when I needed them the most, and my mother Vivian, whose intense love, determination, and...

...Mom, there are no words to describe just how important you are to me.

ACKNOWLEDGEMENTS

Many thanks go to Dr. Wuxu Peng, whom without his guidance and inspiration I would not be involved in research. I would like to thank Dr. Xiao Chen and Dr. Stan McClellan for their support during the thesis process. I also wish to thank my loving family for their patience and understanding during many late nights of study for the past few years.

Finally, I would like to thank the people dear to my heart: Hilliard and Tanya Robertson, Shawn and Sarah Croft, Sam and Sarah Brown, Nicholas and Beth Shaff, and Eric and Pamela Brown, and all of their families, whose love and support I have relied upon all of these years, and for many years to come.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure **Page**

ABSTRACT

Cloud computing is a relatively new idea where virtualized computers communicating with each other perform a multitude of services such as general purpose computing, cloud services to local applications for storage or processing power, and hosting services for entire applications executed within the cloud. This thesis focuses on network bandwidth utilization within the cloud-computing environment and proposes the concept of providing bandwidth control as a service (Bandwidth as a Service) within the cloud. Prior methods of providing control over cloud bandwidth utilization are discussed, and a novel algorithm is proposed for dynamically shaping cloud bandwidth into percentage groups containing individual transport-level connections, maximizing bandwidth efficiency while maintaining limitations on network resources. An analysis of inter-virtual machine bandwidth performance is discussed along with a survey of throughput efficiency and improvements within the hypervisor.

I. INTRODUCTION

Cloud computing may be defined as delivery of computing services, where resources and information are provided to users over the cloud network (Pawar and Wagh, 2012). The services offered by cloud computing providers utilize virtualized resources such as storage, service platforms, and computational processing (Vaquero et al., 2008). Cloud hosting companies provide these resources by charging usage-based pricing, where cloud tenants pay for used resources (Luglio et al., 2014). These resources are used for the various different service capabilities of the Cloud, such as Infrastructure as a Service, where tenants run virtual machines to execute customer business logic; Platform as a Service, where tenants are provided an API[1] to develop cloud-based applications; and Software as a Service, where end-user software is provided to the tenant and executed in the cloud (Vaquero et al., 2008; Li et al., 2010). Cloud services are consumed by cloud tenants, and those services are implemented by virtual machines running within physical hosts. Tenants may or may not be aware of the presence of the virtual machines, depending on the cloud platform, however virtual machines are necessary to carry out the provided cloud service on behalf of the tenant. In this context, cloud tenants may either be end users (humans), or virtual machines also within the cloud. There are many physical host machines within a cloud data center, all of which serve multiple tenants and provide many cloud-based services.

[1] Application Program Interface

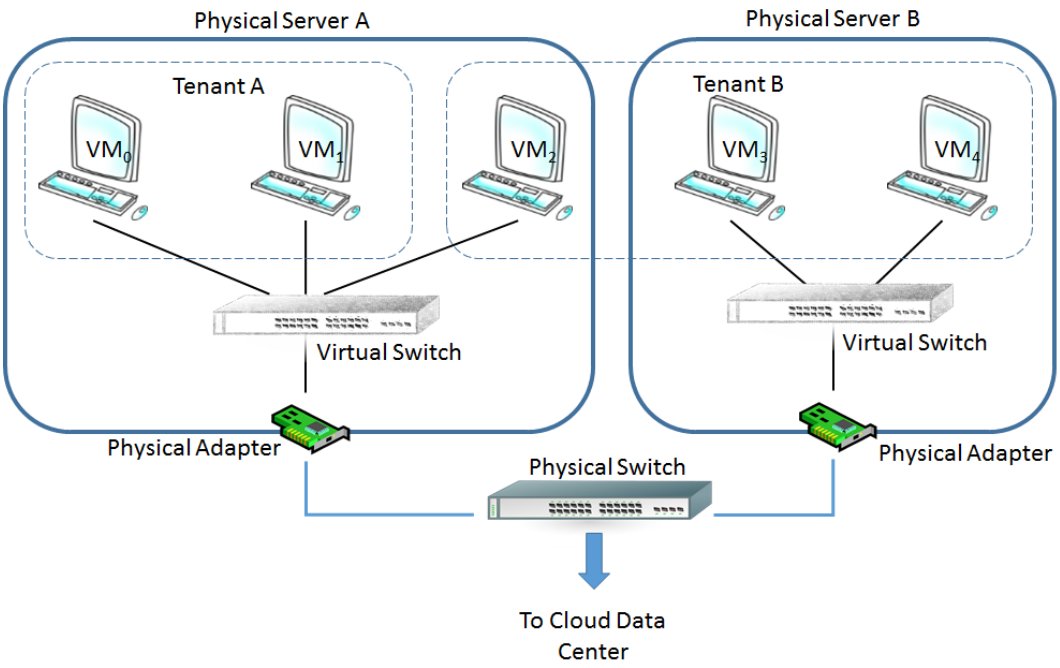


Figure I.1: Tenants and Servers within the Cloud Data Center

Motivation

While virtual machine resources such as memory, processing power, and storage space may be specified by the business arrangement between tenants and cloud providers, the underlying physical network is shared by all tenants and respective virtual machines (see Figure I.1). Due to many tenants utilizing the physical network without regulation, network utilization by some tenants may negatively impact available network bandwidth for other tenants within the cloud. As a result, bandwidth utilization in the cloud may be unpredictable, based on the utilization of network resources by cloud tenants and their virtual machines (Ballani et al., 2013). Additionally, the unpredictable nature of tenant virtual machines usage of network resources allows the possibility of a few VM's[2] disproportionately affecting network throughput of other tenants (Shieh et al., 2011).

Consider the two primary classifications of network bandwidth utilization within the cloud: frequent short transfers (commands, queries) and bulk data transfers (file backup, data mining) (Luglio et al., 2014). If aggregate cloud bandwidth uti-

[2]Virtual Machines

lization is dominated by bulk data transfers then that leaves little to no bandwidth for frequent short transfers, or vice-versa.

Static bandwidth limits do not take into account the bandwidth requirements by the connections hosted within the cloud, which lead to inefficient utilization of bandwidth resources (Popa et al., 2013). This represents a need to manage network bandwidth consumed by each virtual machine to prevent contestation and congestion of physical network bandwidth resources.

Bandwidth as a Service

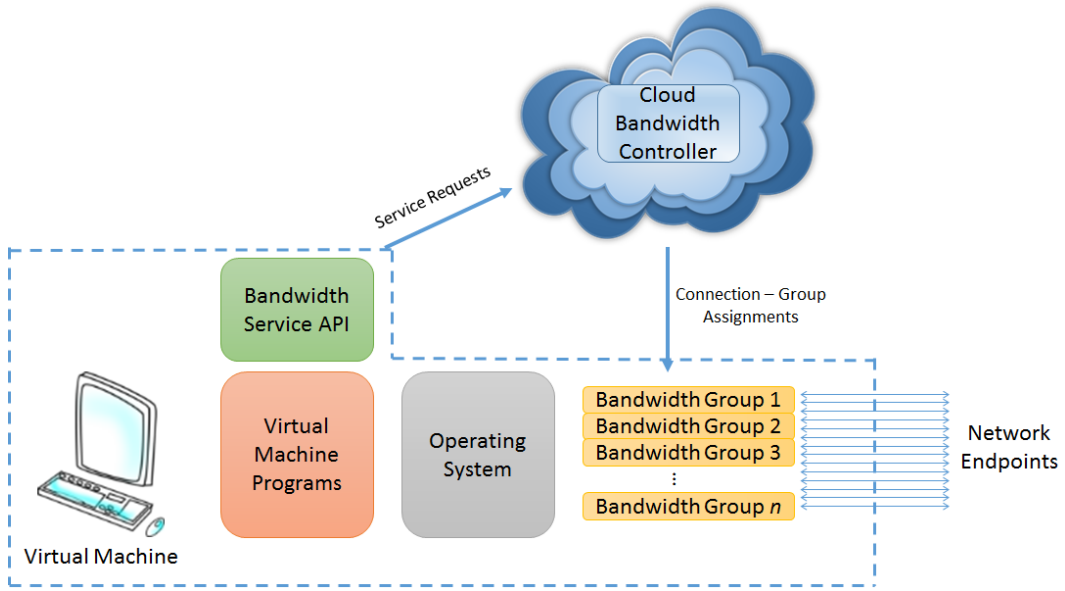
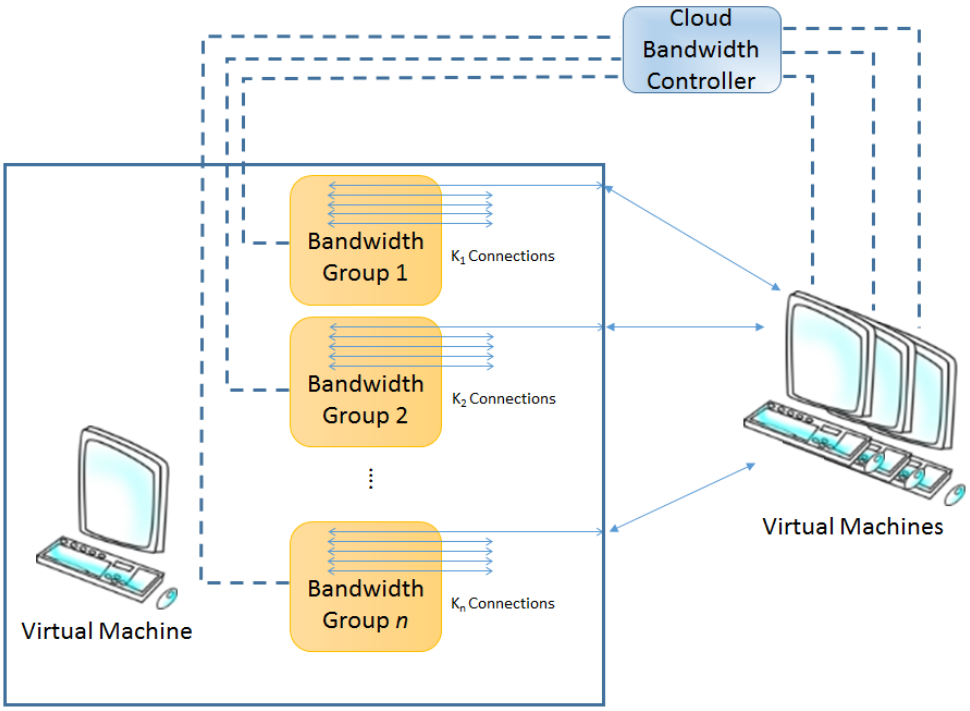


Figure I.2: Bandwidth as a Service - Conceptual View

This work proposes bandwidth management to occur at the virtual machine connection level of granularity. Groups of network connections are assigned percentages of virtual machine bandwidth, marshaled by the cloud bandwidth controller. A bandwidth service API for manipulation of bandwidth guarantees is provided to programs running on the virtual machine. These features represent the foundation of "Bandwidth as a Service". As shown in Figure I.2, tenants may specify percentages of bandwidth guaranteed to groups of network connections by the cloud bandwidth controller. Enforcement of bandwidth guarantees occurs at virtual machine operating system level at the direction of the cloud bandwidth controller.

This provides the bandwidth controller information about bandwidth utilization for each virtual machine, allowing for dynamic bandwidth management.



$$BG(out)_1 + BG(out)_2 + \cdots + BG(out)_n = 100\%\ Outbound\ Bandwidth\ Limit$$
$$BG(in)_1 + BG(in)_2 + \cdots + BG(in)_n = 100\%\ Inbound\ Bandwidth\ Limit$$

Figure I.3: Dynamic Management of Bandwidth Groups

We propose a solution whereby the cloud bandwidth controller assigns a fixed bandwidth limit per virtual machine for both upstream and downstream traffic. At the request of tenant programs, the bandwidth controller may divide virtual machine bandwidth into $BG(in)_n$ and $BG(out)_n$ bandwidth groups. As show in Figure I.3, each group represents a minimum assurance best effort bandwidth as a percentage of total bandwidth for incoming and outgoing traffic. The tenant may assign k connections (unbounded) to a bandwidth group for those connections to be assured a minimum percentage of the available bandwidth. Connections within a bandwidth group share the available bandwidth in accordance with the VM operating system network connection algorithm.

$$BG(out)_1 + BG(out)_2 + \cdots + BG(out)_n = 100\%\ Outbound\ Bandwidth$$

$$BG(in)_1 + BG(in)_2 + \cdots + BG(in)_n = 100\%\ Inbound\ Bandwidth$$

$$k_{connections} \in (-\infty, \infty),\ per\ bandwidth\ group$$

$$n \in (-\infty, \infty),\ for\ n\ groups$$

In this fashion, tenant VMs may ensure particular connections are apportioned bandwidth proportional to the total bandwidth allotted to the virtual machine. While the tenant virtual machines are subject to service level agreements (SLA's) that dictate the total amount of bandwidth available to the VM, connection grouping allows the VM to explicitly instruct the bandwidth controller to choose which connections packets are dropped in the event that bandwidth limitations are reached, ensuring best-effort bandwidth guarantees. Additionally, the programs running within the tenant VM may access the bandwidth service API to programmatically request *in situ* modification to the bandwidth groups and network connection assignments. This gives the tenant the ability to control how bandwidth limitations are applied to network connections directly by virtual machine programming, allowing it to respond to demands for bandwidth control by business logic.

Thesis Structure

This thesis is organized in eight chapters. Chapter I introduces the primary motivation of this research and discusses the concept of Bandwidth as a Service. Chapter II compares and contrasts related work and other approaches against our research. Chapter III discusses the general concept of bandwidth control, discussing the behavior of connections contained within bandwidth groups and how bandwidth is controlled at the virtual machine level. Chapter IV discusses the algorithm used for bandwidth control, the quanta timer mechanism and the description of the implementation. Chapter V reviews the experimental data which

confirms our assumptions and design goals, including a description of the test setup and results. Chapter VI makes a comparison between the network bridge solution included with the Xen Hypervisor and alternative technologies such as Open vSwitch. Discussion of other technologies follows for future analysis and application to this work. Chapter VII discusses abnormalities discovered during the experimentation phase of our research, such as covetous bandwidth effect, expectations of the SCTP protocol in response to dropped packets and the SCTP streaming concept. Chapter VIII concludes with an overall discussion of this work and the direction of future work in Bandwidth as a Service research. Appendices enumerate all data and source code utilized for the findings of this research. Finally, we list all references and citations supporting this thesis.

II. RELATED WORK

CloudMirror: Application-Aware Bandwidth Reservations in the Cloud

Cloud providers typically do not provide strong bandwidth guarantees (defined in service level agreements) for virtual machines in the cloud network. Cloud applications (such as MapReduce) often have tight timing requirements in addition to bandwidth requirements. Since bandwidth is effectively shared by virtual machines within the cloud, highly bandwidth intensive applications can negatively impact the network for other cloud tenants and their bandwidth-dependent cloud applications. Lee, *et al*, state that the current abstraction for bandwidth guarantees which models cloud network topology is insufficient, if straightforward. They claim that cloud application communications do not have similarities with the physical network topology, causing inefficient utilization of bandwidth resources (Lee et al., 2013).

Lee, *et al*, believe the solution lies in redefining the bandwidth guarantee abstraction around the concept of a *cloud application*. They propose CloudMirror, the use of tenant application graphs to describe bandwidth requirements at the cloud application level using network abstraction models against the application communication structure, as opposed to the physical network topology. They give an example of a tiered application structure, namely web, business logic, and database. The communications relationships between each tier are described by the application graph. Virtual machines are assigned to implement each tier with guaranteed bandwidths specified by the tenants in the graph, providing a simplified model of bandwidth guarantees and allocation (Lee et al., 2013).

Their simulation model is based on the service workload encountered by Microsoft *bing.com* data center. Taking this workload simulation they applied the tenant application graph modeling and virtual machine placement algorithm to the simulated workload and compared against Oktopus (Ballani et al., 2011), citing

a sixty percent reduction of network resource utilization than tenant application groups. In terms of the number of virtual machines deployed, tenant application graphs may instantiate up to fifteen times more virtual machines than Oktopus, with the performance test stopping at the first virtual machine instantiation failure (Lee et al., 2013).

This work presents the bandwidth abstraction at the application level, while intuitive, does not provide the required granularity of connection level management of bandwidth. Additionally this method does not allow for real-time modifications to bandwidth guarantees as business logic dictates (Lee et al., 2013).

Sharing Bandwidth by Allocating Switch Buffer in Data Center Networks

Due to fast round trip times, network switches within the data center network frequently do not have full utilization of allocated packet buffers used for network switching operations. Large amounts of small messages combined with the "churn" that is caused by the instantiation of virtual machines and addition of tenants (Shieh et al., 2011) contribute to switch buffer sizes being larger than the amount of traffic floes through the data center network. Additional challenges include TCP handling short flows whose lifetime is less than one round trip time, which results in serious under-utilization of network bandwidth (Kartik, 2003); the TCP Incast problem where multiple endpoints send data packets to a single endpoint, causing short increases in bandwidth utilization leading to increased retransmission timeout periods; and the TCP Outcast problem where two TCP flows arrive at two inputs of an endpoint, and those flows are forwarded through a single output port causing the smaller flow to have lower throughput (Zhang et al., 2014). Zhang, *et al*, view these data center network goodput challenges ultimately as a product of TCP bandwidth waste during timeout periods (Zhang et al., 2014).

Their approach focuses on modifying the TCP transport protocol to form a new flow control mechanism (SAB) which allocates switch buffer sizes appropriate to

adjust the congestion window of each data flow to their optimum values, minimizing packet losses within data center networks. Advantages to this model include fast convergence of transport congestion window sizes to their optimum values, and a low rate of dropped packets. Since packet losses are at a minimum with this approach, the bandwidth inefficiency of TCP Incast and the non-homogenous bandwidth forwarding associated with TCP Outcast are minimized as well. Senders set the TCP window field to *0xffff*, and when they receive the associated ACK, the TCP window field in the ACK header the transport sets the congestion window for future transmissions. The switches will compute the congestion window as traffic flows in real time, and modifies the window field of each data packet passing through. The receiver, when crafting the ACK will place the minimum of either the switch assigned congestion window or the advertised congestion window of the endpoint in the window field, and send it to the sender (Zhang et al., 2014).

Zhang, *et al*, use NetFPGA and the ns-2 network simulator platform to show their assertions of fast congestion window convergence are true as well as prove the scalability of their solution with additional bandwidth and network actors. Their test results focus on query floes and background floes, compared against DCTCP, D2TCP and Reno TCP. They show in the 99.9th percentile of query floe traffic, query floe completion time for SAB is on par with D2TCP and DCTCP, and all three are order of magnitude better than TCP (due to 200 millisecond timeout defaults). In the 99.99th percentile of query floe traffic, D2TCP and DCTCP resulted in query flow completion times of ten milliseconds on average, while SAB resulted in approximately two milliseconds, on average. For 95th percentile, and 99th percentile, the average query floe completion time had no clear improvement between any of the profiled transport mechanisms (Zhang et al., 2014).

Their approach focuses on modifying the transport protocol to allocate switch buffer sizes appropriate to the congestion window of each data flow, minimizing packet losses within data center networks. This approach assumes a single transport protocol within the data center and ignores other protocol types, such as

UDP. This is a key disadvantage as Shieh, *et al*, specify that a cloud based virtual machine may arbitrarily run any number of applications, use any networking protocol, use any number of data flows, or choose networking endpoints with which to communicate (Shieh et al., 2011). Additionally the scope of this solution does not address tenant specific bandwidth guarantees (Zhang et al., 2014).

Sharing the Data Center Network

The shared nature of the data center networks requires allocation mechanisms for CPU, memory and disk resources to data center tenants, however allocation mechanisms for tenant network resources are not satisfactory. Mechanisms such as TCP and it's variants help to scale traffic loads to achieve network fairness amongst other tenants, however this is unsatisfactory to impose isolation between tenants. Since the network is a shared resource between tenants, network application utilization of network bandwidth may negatively impact other tenant network applications. Shieh, *et al*, also comment on how unpredictable virtual machine applications are with respect to network utilization since tenants have free reign on the number of network connections and congestion control mechanisms (or lack thereof) to utilize within the shared network. (Shieh et al., 2011).

Shieh, *et al*, address this lack of performance isolation by introducing Seawall, a logical abstraction whereby endpoints are assigned a weight factor, excluding tenants or groups of virtual machines. This weight factor controls the amount of bandwidth available on links between endpoints such that bandwidth allocation is limited by the smallest share of bandwidth on links between endpoints. The bandwidth allocator for Seawall considers the weight factor of each endpoint along with feedback statistics on congestion control parameters of each entity which the endpoint communicates, and generates allowed throughput rates for each endpoint. At the virtual machine level, the congestion control mechanism is replaced by a query mechanism to a rate limiter object that has knowledge of how much bandwidth is allotted for this endpoint, and bandwidth is limited in this fashion.

Seawall uses hypervisor IPC to report the congestion window maximum to each virtual machine so that traffic can flow to the rate limiter in an efficient manner. This approach is independent of network utilization of other tenants since the data center administrator is capable of specifying how much bandwidth traffic sources receive (Shieh et al., 2011).

The Seawall prototype is an NDIS packet filter architecture which sits between the TCP/IP stack and the Ethernet card driver, but could also be employed at the virtual switch level of the hypervisor. Shieh, *et al* note that technologies such as SR-IOV which allow virtual machines to bypass the hypervisor can defeat Seawall, and suggests hardware-offload techniques should implement support for Seawall. Benchmarks were conducted using the NDIS filter, where a marginal amount of overhead (3.8 percent at sender and 3.4 percent at receiver) accrues due to the additional shim filter. Most of the additional overhead is the shim filter; a null NDIS filter itself adds 1.8 percent at the sender and 2.7 percent at the receiver. Because of this, performance of this implementation was deemed to scale well in a data center (Shieh et al., 2011).

We find that their work does not allow for the tenant to specify bandwidth allocations for their applications network utilization profile, opting in favor of the tenant choosing classes of network service throughput (Shieh et al., 2011). We believe that this approach does not provide the tenant the granularity required for dynamically assigning bandwidth allocations to specific connections within cloud-based tenant applications; a key point since Shieh, *et al*, comment on how unpredictable virtual machine applications are with respect to network utilization (Shieh et al., 2011).

Towards Predictable Data Center Networks

The unpredictability of network performance within the cloud is contributed to by a many factors. These factors include the lack of cloud network resource guarantees on a per-tenant basis, which causes variability in cloud application performance.

This lack of cloud network performance guarantees inhibits the cloud from supporting various application requirements unless great cost is expended to make up for the variability of network performance. Since poor network performance within the cloud can translate to slow completion of cloud application tasks, it can mean increased cost for the cloud tenant and their clients (Ballani et al., 2011).

Ballani, Costa *et al* discuss OKTOPUS, a system which utilizes tenant specific virtual networks connecting virtual machines utilized by the tenant. This work proposes two network abstractions. A *Virtual Cluster* is an abstraction which imagines all virtual machines are connected to a non-oversubscribed virtual switch. A *Virtual Oversubscribed Cluster* is where groups of virtual machines are connected to a group-specific virtual switch, and each group-specific virtual switch is connected to a root virtual switch. This allows for improved bandwidth between communications within a specific group, at the expense of bandwidth for communications between groups. The root virtual switch maximum bandwidth is approximately the number of virtual machines multiplied by the maximum bandwidth of each virtual machine[1]. The tenant will have the choice of not only the virtual machines, but also a choice of the tenant network structure, such that if the application requires better communication performance between tenant virtual machines, they could be allocated to a group virtual switch. This virtual switch would then be connected to the rest of the data center (Internet) through the root virtual switch. This allows greater performance for communications between tenant virtual machine instances in the group at the expense of worse communications performance outside the group (Ballani et al., 2011).

For job completion times of MapReduce operations, *Virtual Clusters* provided measurable and substantial improvement over a traditional baseline measurement of current cloud virtual machine allocation techniques. The *Virtual Oversubscribed Clusters* mechanism also showed improvement over the baseline, taking into account that the benefit for increased over-subscription layers will reduce over time.

[1] assuming maximum bandwidth capability is homogenous across VMs

The primary reason stated by Ballani, Costa *et al* for the increased improvement of *Virtual Oversubscribed Clusters* when compared to the baseline is that the virtual machines in the baseline are not allocated with consideration to the application bandwidth requirements, causing the application execution to be blocked waiting on inefficient network communications topology. They also assert that even without over-subscription, there is a 2X performance improvement between the *Virtual Cluster* topology and the baseline (Ballani et al., 2011).

Their work assumes that each cluster of virtual machines is homogenous regarding application or purpose; a view adopted from the concept of application dedicated data centers (Ballani et al., 2011). We believe that cloud computing has a varying field of multiple application types and bandwidth utilization profiles as described by Shieh, *et al*, and that it is possible to increase granularity of bandwidth control beyond the cluster, or virtual machine level (Ballani et al., 2011).

Chatty Tenants and the Cloud Network Sharing Problem

Cloud tenants encounter variable network performance which impacts application performance and tenant cost. Offering minimum bandwidth guarantees is considered a naive solution since the problem of bandwidth limitation enforcement becomes more complex as the number of tenants and their instantiated virtual machines increase within the cloud networking environment. Additionally, the question of cost proportionality rises if communication is between inter-virtual machine instances by separate tenants. The focus for improvement is based on the observation that upwards of thirty-five to forty percent of utilized bandwidth is between tenants within the cloud, additional efforts should concentrate on this networking characteristic for the greatest performance impact (Ballani et al., 2013).

As such, Ballani, Jang, *et al* discuss Hadrian, a bandwidth allocation framework for multi-tenant cloud environments. Their work focuses on providing minimum bandwidth guarantees for inter-tenant communication only, where virtual ma-

chines declare communication dependencies with each other upfront. Bandwidth allocations are enforced at the switch level to achieve hose-model compliance[2]. Additionally, Hadrian makes use of the guaranteed bandwidth limit configurations between tenant virtual machines and their virtual machine endpoints to drive a more efficient virtual machine placement framework (Ballani et al., 2013).

The assertions by Ballani, Jang, *et al* are tested with direct experiments and with simulations where a cloud workload is generated and is comprised of tenant requests for instantiating virtual machines. For the experiments, the baseline measurement had virtual machines placed in a greedy fashion in proximity to other virtual machines for the tenant. A second baseline is also recorded where the virtual machines were placed in a greedy fashion in proximity to virtual machines for which there was a communication dependency. The third metric is the Hadrian framework. The first baseline accepted thirty-five percent of virtual machine instantiation requests, while the second baseline accepted fifty-five percent of requests. Hadrian accepted seventy-two percent of requests, demonstrating a more efficient mechanism for utilizing communications dependency to drive virtual machine instantiation and placement. Their simulations represent similar findings where Hadrian can accept twenty percent more requests than both the first and second baseline comparisons (Ballani et al., 2013).

The "hose model" is where all virtual machine bandwidth is regulated through virtual switches that have a dedicated network link (hose) with a minimum bandwidth guarantee. The hose model offers a large grain solution, sacrificing efficiency at the virtual switch level; making no distinction between virtual machines connected to the virtual switch (Lee et al., 2014). While their work shows promise in their experimental results, it is limited to inter-tenant communications; this presents a very limited focus for a generic solution. Additionally, Ballani *et al* note that the expectation to know tenant communication dependencies beforehand is

[2]The "hose-model" is where all virtual machine bandwidth is regulated through virtual switches that have a dedicated network link (hose) with a minimum bandwidth guarantee. The hose model offers a large grain solution, sacrificing efficiency at the virtual switch level; making no distinction between virtual machines connected to the virtual switch (Lee et al., 2014)

challenging (Ballani et al., 2013).

Gatekeeper: Bandwidth Guarantees for Multi-Tenant Datacenter Networks

Cloud providers are required to provide isolated performance guarantees for virtual machines within the data center. Applications which require the transporting large amounts of data across the network depend on these guarantees. Rodrigues, *et al* believe that network performance isolation solutions must respect four requirements to be effective. Scalability of the solution to allow for large amounts of virtual machines in communication with each other at high speeds and high rates of churn[3]. The solution should provide explicit levels of performance to tenants when virtual machines are deployed, to gauge the optimal point of network performance and tenant cost. The solution should provide protection against malicious virtual machine instances which, taking advantage of the shared nature of cloud networking, may intentionally (or non-intentionally) attempt to consume unnecessary bandwidth at the expense of other tenant virtual machine instances. Finally, the solution should not depend on deterministic guarantees since these tend to resort to over-reservation of bandwidth resources to guarantee a minimum bandwidth allocation, thus causing inefficiencies in bandwidth allocation and utilization. Instead, the bandwidth guarantee service should provide both minimum and maximum limits to give tenants a better expectation of the window of performance supported by the cloud networking environment (Rodrigues et al., 2011).

Rodrigues, *et al* discuss Gatekeeper, which utilizes the virtualization layer of the data center server to limit bandwidth using guaranteed allocations. Bandwidth guarantees are provided for ingress and egress traffic at the physical link, and then minimum guarantees of network bandwidth are allocated to the virtual NICs per virtual machine instance connected to the physical link. If a virtual NIC is not utilizing all of its minimally guaranteed bandwidth, the excess is available for use by other virtual network interfaces or virtual machine instances. Addition-

[3]Churn in this context is defined as the effect on networking bandwidth when virtual machines are instantiated and destroyed frequently

ally, a virtual NIC has a maximum limit to prevent over-utilization and allow for deterministic behavior. Gatekeeper utilizes something similar to active congestion notification, whereby if the receiver bandwidth is being exceeded, a message is sent to other servers that the receiver traffic is in excess. In the event a virtual NIC is utilizing excess bandwidth beyond its minimum guaranteed bandwidth and it is explicitly notified that their rate is being exceeded, the virtual NIC is set to its minimum guarantee. Gatekeeper monitors bandwidth at a period of ten milliseconds, but is configurable (Rodrigues et al., 2011).

Gatekeeper implementation is evaluated with five servers connected to a one Gigabit/second ethernet switch. One server has two virtual machine instances, and the remaining four servers have one virtual machine each. Their experiments show that tenants using TCP respond well to straight bandwidth capping, but cannot account for tenants that generate traffic that do not require acknowledgement. It is noted that bandwidth capping mechanisms cannot repurpose reserved bandwidth which remains unused by the tenants. Gatekeeper is able to achieve both of these shortcomings for both egress and ingress traffic. When profiled on Xen hypervisor the Dom0 CPU load showed an increase of ten percent (Rodrigues et al., 2011).

The authors note that this rate decrease can cause underutilization of the link and leave that analysis open to future work (Rodrigues et al., 2011). We believe this work is promising, however lacks granularity regarding the individual connections created by the virtual machines within the cloud.

FairCloud: Sharing the Network in Cloud Computing

The shared network experienced in the cloud environment represents a problem of performance isolation for virtual machines. Popa, *et al* layout three guidelines for a desirable solution to this issue. First, each tenant should have an expectation of minimum guaranteed bandwidth, purchased by the tenant and irrespective of the network utilization of other tenant virtual machines. Second, network resources should be efficiently utilized when reserved resources are not under demand load.

This implies that minimum bandwidth guarantees not fully utilized by one tenant can be made available to another tenant for consumption; this can be important for bursty traffic behaviors. Third, network resource allocation should be allocated based on what the tenant is willing to pay for, similar to CPU and memory resources within the cloud. The challenges to achieving these requirements stem from the inverse relationship of minimum bandwidth guarantees and network proportionality[4]. A similar challenge also exists between providing high utilization of network resources and network proportionality (Popa et al., 2012).

Popa, *et al* discuss FairCloud, a set of cloud based network sharing policies in an attempt to fairly distribute network bandwidth as a resource fairly across all virtual machines hosted for each tenant within the cloud. Three bandwidth allocation policies are discussed. *Proportional Sharing at Link-Level* is the weighted bandwidth allocation at the switch level providing bandwidth to virtual machines that communicate on a particular switch-link, with weighted fair queuing per tenant. *Proportional Sharing at Network-Level* is a bandwidth sharing policy such that virtual machines communicating with each other has the "same total weight through the network", regardless of the actual network topography of virtual machines within the tenant's control. It is interesting to note that Popa, *et al* concede that detailed information about the network load at link bottlenecks is not known, and they assume uniform load conditions across the entire network. *Proportional Sharing on Proximate Links* is where network links are prioritized based on the "importance" of the network link and the virtual machines utilizing that link (Popa et al., 2012).

The performance evaluation shows that *Proportional Sharing on Proximate Links* provides the best bandwidth guarantees while *Proportional Sharing at Network-Level* provides the best network proportionality. *Proportional Sharing at Network-Level* achieves near-network proportionality for MapReduce requests. *Proportional*

[4]network proportionality is considered the distribution of network bandwidth to tenants *in proportion to* the number of virtual machines contained by each tenant, in addition to bandwidth paid for between tenants (Mogul and Popa, 2012; Popa et al., 2012)

Sharing at Network-Level and *Proportional Sharing on Proximate Links* both positively impact bandwidth allocations for application performance improvements up to fifteen times. For *Proportional Sharing at Link-Level*, evaluations with MapReduce jobs show that proportional bandwidth allocation is not directly dependent on the number of mappers (Popa et al., 2012).

Network level proportional sharing is a proposal that operates "in the absence of detailed knowledge about the load on each bottleneck link, assuming uniform load conditions throughout the network" (Popa et al., 2012). This is an assumption that existing research literature has contradicted in favor of non-uniform load conditions within the cloud (Shieh et al., 2011). The policy of proximate sharing to the network link is an unfair proportion bandwidth policy because it assigns importance to virtual machines using a specific network link within the cloud data center and less importance on the remote machine with which it communicates. This can cause unexpected failures by tenant applications written expecting a fair allocation of bandwidth between source and destination network endpoints.

SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees

Service Level Agreements define the tenant cost and availability for bandwidth, CPU utilization, and storage. Yet, SLAs do not define the cost or availabiility for bandwitdh guarantees between virtual machines within the cloud (Guo et al., 2010).

Guo *et al*, propose a new form of virtual machine abstraction, a "virtual data center" which comprises virtual machines within the cloud that represent a logical grouping, such as virtual machines that are within tenant control, or subject to service level agreements. Their cloud architecture, called SecondNet, aims to achieve the following goals. *Scalability*, with potentially millions of virtual machines running within the cloud. *High utilization*, where bandwidth is utilized to its fullest extent with little inefficiencies. *Elasticity*, an ability to cope when tenant

requirements change. Finally, *bandwidth guarantees* for inter-virtual machine communication in addition to the aforementioned goals. Bandwidth control is implemented at the hypervisor level as their research claims that switch level bandwidth control is prohibitively expensive; this moves forward their goal of *scalability*. To support *elasticity*, SecondNet utilizes virtual machine migration between server clusters, such that virtual machines and their aggregate virtual data centers, are on the same physical host or physically interconnected hosts. SecondNet utilizes a *VDC Manager* to handle tenant requests for bandwidth and virtual machine instantiation (Guo et al., 2010).

Since SecondNet is primarily implemented at the hypervisor level, it can make use of off-the-shelf equipment. The simulation performed by Guo *et al*, comprise a virtual data center allocated with five thousand virtual machines. Their experimental results prove their accomplishment of *high utilization* where server bandwidth is utilized at ninety percent of network resource capacity (Guo et al., 2010).

We feel that this method of bandwidth provision is too large grained to satisfy tenant expectations of bandwidth guarantees. Additionally, there seems to be no proposal for capturing unused bandwidth indicative of non-uniform load conditions within the cloud (Shieh et al., 2011).

Towards Flexible Guarantees in Clouds: Adaptive Bandwidth Allocation and Pricing

Bandwidth sharing is an issue that is addressed by the Internet at large. Similarly cloud computing shares the same problem space albeit a smaller subset. Since cloud data centers are typically managed by a single entity, we can take advantage of the full knowledge of every network path and fixed link within (Divakaran and Gurusamy, 2015).

Divakaran, *et al* discuss two forms of bandwidth reservation, bandwidth requests for constant allocation of bandwidth, and time guarantee requests for data

transfers within a specific guaranteed time frame. Bandwidth profiles per tenant are estimated by measuring bandwidth utilization on an hourly basis. These bandwidth profiles are fed into their bandwidth controller and granted to tenants based on demand. Their bandwidth allocator first is designed to maximize revenue by allocating only the minimum bandwidth necessary for each request, and secondly will allocate more bandwidth based on the previous measurement of prior bandwidth utilization in an attempt to predict demands on the network infrastructure by the tenant. Separately, small amounts of bandwidth are reserved for infrequent bursts of high utilization and background transfers (Divakaran and Gurusamy, 2015).

Evaluations consist of comparing their *adaptive flexible bandwidth allocator* (A-FBA) algorithm to the commonly used *deterministic bandwidth allocator*, which simply enforces strict bandwidth limitations. Two investigations of performance metrics are discussed, a straight comparison and a comparison that includes tenant cost. In the straight comparison, A-FBA reduces rejection of bandwidth allocation requests by fifty percent; in some cases by more than eighty percent for certain load scenarios. In the comparison including differential cost[5] A-FBA allocated eighteen to twenty-six percent more bandwidth as compared to DBA. Revenue went up by twelve percent on average, with a maximum price discount of twenty percent. Revenue could be maximized by decreasing the discount rate (Divakaran and Gurusamy, 2015).

We find this approach is not deterministic on immediate demands, and makes an assumption that bandwidth utilization can be predicted based on prior bandwidth utilization. Shieh, *et al* show how non-uniform load conditions within the cloud contradict the notion that bandwidth utilization may be predicted (Shieh et al., 2011).

[5]In this case, a difference in price for bandwidth-guarantee and time-guarantee reservations

ElasticSwitch: Practical Work-Conserving Bandwidth Guarantees for Cloud Computing

Since cloud network resources are shared between tenant virtual machines, there is no way to determine worst case performance for applications running in the cloud network environment. Static bandwidth guarantees exist, however they are not *work-conserving*[6]. A cloud network environment should be *work-conserving*, deployable, scalable, and provide minimum bandwidth guarantees (Popa et al., 2013).

Popa, Yalagandula, *et al* discuss ElasticSwitch, a mechanism to provide bandwidth partitioning according to the hose-model with the added benefit of being work-conserving; where bandwidth unused by one virtual machine is available for use by another virtual machine. They employ guarantee partitioning to ensure hose-model compliance and rate allocation to increase bandwidth utilization when excess bandwidth is available. ElasticSwitch is a decentralized bandwidth control system, intentionally lacking a centralized knowledge-store of all bandwidth enforcement actions taken by the virtual machine hypervisors (Popa et al., 2013).

Evaluation shows that when testing with three hundred virtual machines all in contention to communicate with a single virtual machine, bandwidth guarantees are preserved. The bandwidth guarantees also occur irrespective of congestion location within the network. Since seventy-five to ninety-nine percent of the network links are utilized, ElastiSwitch is considered *work-conserving*. Explicit congestion notification also improves network efficiency, if available with the commodity networking equipment as prescribed in their goal to provide a practical solution (Popa et al., 2013).

This work is the closest to the solution we discuss as part of our work, however with clear distinctions. We propose a service-oriented bandwidth control mechanism that is centralized while ElasticSwitch is designed to be distributed (Popa

[6]Bandwidth reserved for one tenant's virtual machines may not be utilized for another tenant, regardless if the bandwidth is being utilized

et al., 2013). This causes issues where the bandwidth utilization of the cloud as a whole cannot be determined through direct measurements or configured from a centralized location. While Popa, Yalagandula, *et al* have valid concerns regarding scalability of a centralized bandwidth control solution (Popa et al., 2013), the overhead associated with centralized bandwidth control is quickly mitigated thanks to advances in data center processing power. Lastly ElasticSwitch operates by knowing which pairs of virtual machines are in communication (Popa et al., 2013), but does not take into account the connection-level granularity of the communications.

BwMan: Bandwidth Manager for Elastic Services in the Cloud

Since network bandwidth within the cloud environment is typically not shared as a managed resource, service level agreements may be violated since physical servers, communication threads and virtual machines all contend for network resources (Liu et al., 2014).

Liu, *et al* describe BwMan, a related solution for bandwidth management by proposing a cloud bandwidth manager that focuses on machine learning models to predict bandwidth demand, thereby reducing violations of service level objectives. The learning algorithm is implemented by a centralized bandwidth controller containing a control loop monitoring the network and dynamically allocating bandwidth quotas according to changing workloads. BwMan implements fine grain control at the network port level, to allow specific network services to have minimum bandwidth guarantees. Liu, *et al* create two separate machine learning models of bandwidth utilization, user-centric and system-centric. The user-centric model correlates performance metrics for user workloads against available bandwidth. Similarly, the system-centric model correlates performance metrics for system workloads against available bandwidth (Liu et al., 2014).

The evaluation of BwMan is performed within a cloud network environment supporting the cloud-based OpenStack Swift storage solution. BwMan prevents

upwards of fifty percent of SLO[7] violations for the Swift distributed storage solution on average. For the user-centric performance measurement, all bandwidth allocations succeeded in providing the prescribed service level objectives. For the system-centric performance measurement, metrics are provided where the Swift data integrity feature is triggered using a data corruption simulator. When the data corruption tool activates, the Swift data integrity feature consumed bandwidth which the system-centric model anticipates and allocates bandwidth quotas to meet the expected demand. Overall as the SLO interval of confidence rises, the percentage of SLO violations decrease with BwMan as compared to without (Liu et al., 2014).

While this work certainly is promising, they assume a homogenous set of network services in the cloud whereas our work leaves the bandwidth connection groups within the control of the user, via the bandwidth service we propose. Additionally, their approach is inherently predictive, which leaves bandwidth open to inefficiencies while the approach in our research is considered a work-conserving "on-demand" approach to eliminate SLO violations (Liu et al., 2014).

QoS-Guaranteed Bandwidth Shifting and Redistribution in Mobile Cloud Environment

When a mobile user is connected to the cloud, they are connected to mobile data gateways which are consequently connected to the cloud service provider. The bandwidth requirements and QoS specifications for the mobile user are also applied to the gateway to the cloud environment. Since mobile users change location on a dynamic and unpredictable basis, there is a need for *bandwidth shifting* where the bandwidth allocations and guarantees that existed for one cloud gateway connected to the mobile user shifts to another cloud gateway for the same mobile user (and same cloud service provider) who has changed physical position. A problem now exists where the second cloud gateway may have a lesser bandwidth guarantee

[7]service-level objective

than the first gateway, causing the QoS specification and bandwidth requirements of the mobile user to be violated (Misra et al., 2014).

Misra, *et al* describes QoS guarantees for mobile cloud computing whereby QoS is maintained with physically moving clients (mobile users) changing cloud gateway entry points. Their work focuses on bandwidth shifting and redistribution algorithms at cloud-gateway entry points along with proposing auction-theory to solve the bandwidth distribution problem for mobile cloud computing. Because of the shifting nature of the mobile end-user, their bandwidth allocation problem differs from the traditional issue of providing bandwidth to all gateways to shifting bandwidth resources to which gateway the end user is connected to at any point in time (Misra et al., 2014).

The simulation for the auction based QoS utility maximization algorithm shows the need for bandwidth redistribution, and demonstrate correctness (Misra et al., 2014).

Because of the shifting nature of the mobile end-user, their bandwidth allocation problem differs from the traditional issue of providing bandwidth to all gateways to shifting bandwidth resources to which gateway the end user is connected to at any point in time (Misra et al., 2014). While this method is very applicable to mobile clouds, there is a core assumption difference whereby we do not expect the end-user virtual machine to change entry points (not including migration of virtual machines within the data center).

A Dynamic Bandwidth Allocator for Virtual Machines in a Cloud Environment

Virtual machine operating within the cloud have performance issues which must be addressed. Cloud applications all share the same available network bandwidth, while existing hypervisors (virtual machine monitors) provide static bandwidth allocations (Amamou et al., 2012).

Amamou, *et al* describe a mechanism for dynamically allocating bandwidth for virtual machines at the network driver level. In an effort to satisfy QoS require-

ments, they implement a series of traffic categories, each of which has a minimum and maximum bandwidth limitation fixed according to the Service Level Agreement. Virtual Ethernet interfaces represent traffic categories to the applications within the virtual machine. The existing Xen hypervisor implementation drops packets due to exceeded resource limitations at the hypervisor level, after the packet has been copied from the virtual machine memory region. In their work, packets exceeding the bandwidth limitations are dropped before the packet is copied to the hypervisor layer, representing a performance improvement over the standard implementation (Amamou et al., 2012).

The performance measurement test bed consists of three virtual machines, each with three virtual interfaces with configured minimum bandwidths (for each virtual interface, on each virtual machine) and maximum bandwidths. The test case lasts for three hundred seconds with configured rates of transmission, with a configured readjustment period of 100 milliseconds. System-wide packet loss is rated at 55.87% before dynamic bandwidth allocation. Afterwards, packet loss drops to 16.56% when applying dynamic bandwidth allocation. CPU utilization increases as the readjustment period gets smaller (Amamou et al., 2012).

Their work is similar to this research in that bandwidth enforcement happens at the virtual machine driver level but there are key differences. Our level of bandwidth control granularity is at the connection level with groups of transport connections, while theirs is larger grained with the use of virtual network interfaces. Secondly, our approach is key in regulating bandwidth as a centralized service within the cloud, rather than bandwidth control isolated to the virtual machine. Thirdly, the addition or reduction of bandwidth groups happens in run-time of the virtual machine along with the addition of connections within those bandwidth groups, while traffic categories specified by Amamou, *et al* are implemented as virtual network interfaces which are difficult to modify on the fly, requiring the reset of the virtual machine. Another difference is the amount of control the virtual machine user has on how their applications utilize bandwidth. Since traffic

categories proposed by Amamou, *et al* are enforced by virtual network interfaces (Amamou et al., 2012), the virtual machine user is limited to which virtual interface to use for their data to fit within SLA-defined bandwidth categories. In our proposal for Bandwidth as a Service, the virtual machine user is given the choice of creating as many bandwidth groups necessary with customizable percentages of allocated bandwidth and the ability to assign connections to specific groups, by using the provided API through the transport protocol driver. This affords the user finer-grain control of how their applications utilize available bandwidth in a truly dynamic programming environment.

Quality of Service (QoS) Guaranteed Network Resource Allocation via Software Defined Networking (SDN)

Multimedia applications within the cloud are subject to undesirable and unpredictable network throughput issues. This can cause problems when the multimedia applications within the cloud have QoS requirements by its users unfulfilled due to packet loss and long travel times. This issue is compounded by large quantities of consumers of multimedia applications, causing a large negative impact on achieving QoS requirements for each consumer (Akella and Xiong, 2014).

Akella and Xiong also propose an endpoint to endpoint QoS guarantee for cloud services. This is accomplished by computing the total delay for the service requests, using this information to modify bandwidth allocations to match the service requested. They classify two types of QoS routing algorithms, dynamic routing path selection based on the state of the cloud network, and static routing path selection. Using virtual switches, their work computes end-to-end delay to compute the best network path for a given cloud service, taking into account number of hops, end to end delay and bandwidth. Their work relies on bandwidth guarantees provided to Open vSwitch using Hierarchical Token Based Queuing Technique, and leverages Open vSwitch to dynamically manipulate bandwidth allocations to satisfy QoS requirements by consumers by selecting paths with available bandwidth to service

consumer requests (Akella and Xiong, 2014).

Their measurements showed an increase in the average packets per second, with one test case showing an improvement from 5000 packets per second to 8200 packets per second, when applying the proposed path selection algorithm for TCP traffic. For UDP traffic, they claim a marked improvement from 1500 packets per second to 6000 packets per second (Akella and Xiong, 2014).

Their work relies on bandwidth guarantees provided to Open vSwitch using Hierarchical Token Based Queuing Technique (Akella and Xiong, 2014). Akella and Xiong attempt to provide bandwidth guarantees using the switch model, but do not have a direct approach in controlling the use of bandwidth by virtual machines in the cloud.

Transport Layer Optimization for Cloud Computing Applications via Satellite: TCP Noordwijk+

In contrast to internet service providers which charge for peak utilization of bandwidth, cloud providers charge a fixed rate for total bandwidth utilization. When satellite networks are utilized for cloud traffic, congestion events occur at a much higher frequency. TCP as a transport protocol in during satellite connectivity experiences performance issues due to variable packet delay, which requires complex quality of service configurations to overcome (Luglio et al., 2014).

Lugio, *et al*, propose a new transport layer algorithm to account for these factors on bandwidth. They identify two major classifications of cloud network utilization profiles, higher priority frequent short transfers and lower priority bulk data transfers. Their proposed transport layer TCP Noordwijk+ separates available bandwidth between both network utilization profiles, preserving priority and available bandwidth. This is accomplished by utilizing TCPN+ proxy tunnels between endpoints, or an endpoint and a remove cloud service. Ideally, services and protocols using TCPN+ are not affected, nor require modification for TCPN+. Adaptive round trip time estimation is used for the retry mechanism, which in turn uti-

lizes an exponentially weighted moving average algorithm. Additionally, dynamic acknowledgement dispersion algorithm is used to accomadate burst timing and sizing. When periods of congestion are encountered, an adaptive priority algorithm is employed which allows bandwidth to be ceded to connections in priority order (Luglio et al., 2014).

Simulations with NS-2, modified to reflect satellite connection conditions, show results when TCP is used over a TCPN+ connection, when TCPN+ is competing with internet traffic from a single terminal, and when TCPN+ is competing with internet traffic from multiple terminals. These simulations shows that TCPN+ uses spare capacity unused by higher priority connections, and will prioritize throughput to a minimum guaranteed value, based on priority (Luglio et al., 2014).

Using the transport layer as a bandwidth control mechanism does have advantages. Indeed, Lugio, *et al* focus their work on a specialized version of TCP, whereas we make use of SCTP for bandwidth control at the virtual machine level. However, Lugio, *et al* do not utilize a central bandwidth controller, crucial in establishing a service oriented bandwidth management scheme.

Towards Bandwidth Guarantee in Multi-Tenancy Cloud Computing Networks

Virtual machines, independent of tenant ownership, are instantiated on physical hosts, and are interconnected through the cloud network. The intermediary network devices such as switches and routers do not distinguish between tenant traffic, and hence simultaneously share the cloud network. This causes network performance to vary unpredictably over time for all tenants utilizing the cloud network (Zhu et al., 2012).

Zhu, *et al*, propose a solution similar to that presented by OKTOPUS (Ballani et al., 2011) with improvements to the virtual machine allocation scheme. An analysis of homogeneous bandwidth demand concludes that heterogeneous bandwidth demand is more accurate and realistic model of tenant bandwidth utilization (Zhu et al., 2012). The concept of bandwidth allocation ranges specified for each network

device provides the mechanism to determine the suitability for placement of the virtual machines. Allocation ranges are described for network switches in terms of the number of aggregated links and residual bandwidth. Allocation ranges are also described for physical servers within the cloud network in terms of the number of available slots to instantiate virtual machines along with utilized bandwidth of the server (Zhu et al., 2012).

Using a small datacenter testbed, experiments are conducted to verify that the work of Zhu, *et al* satisfies per-virtual machine bandwidth demands and provides a virtual machine placement strategy for tenants to maximize cloud network bandwidth. Results reflect reduced task completion times, especially when there are different bandwidth guarantee requests per virtual machine allocation. This shows a marked improvement over the current methodology of allowing virtual machines to compete against each other for available cloud network bandwidth (Zhu et al., 2012).

Zhu, *et al*, do not provide a centralized concept of bandwidth control, and the granularity of network traffic control is at the virtual machine level with no mechanism for the user to classify individual connections (Zhu et al., 2012).

NetShare and Stochastic NetShare: Predictable Bandwidth Allocation for Data Centers

Cloud network performance can greatly impact cloud application performance, while stringent service level agreements are put in place by tenants to ensure proper cloud application performance. Static bandwidth limitations can enforce these performance requirements by cloud applications but result in an inefficient use of cloud network bandwidth in the presence of bursty cloud network utilization. Over-provisioning is recognized as a potential solution, however the increased cost for additional bandwidth to guarantee all tenant SLA bandwidth requirements is too costly as a general solution (Radhakrishnan et al., 2012).

Lam, *et al*, propose NetShare, a bandwidth allocation mechanism which services

are weighted for prioritization of cloud network resources with maximum and minimum bandwidth limitations. Cloud services are assigned specific weights which are applied as a bandwidth prioritization scheme for maximum bandwidth provisioning. Lam, *et al*, note that minimum bandwidth guarantees are not available in the current implementation of NetShare, but could be added using a virtual machine placement algorithm that took into account available bandwidth to guarantee minimum bandwidth requirements. They discuss a mechanism for automatic weight assignment by taking into account switch port utilization and virtual machine placement with some assumptions on network tree structure. They also discuss utilizing TCP for bandwidth enforcement by adjusting the sliding window for flow control, a throttling mechanism for *connection-less* transport protocols such as UDP, and their implementation of a centralized bandwidth allocator (Radhakrishnan et al., 2012).

Cloud application completion times are the primary performance metric for experiments showing the effectiveness of NetShare. Hadoop sort jobs were executed, with NetShare providing twice the application performance, cutting application completion times in half, on average. Group allocation of TCP flows and rate throttling of UDP links show a higher bandwidth utilization combined with lower instances of packet drop (Radhakrishnan et al., 2012).

The authors work is missing minimum bandwidth guarantees, but also automatic weight distribution can be problematic since the bandwidth weight assignment algorithm would have to be customized for each cloud provider's specific use case. Their use of transport level throughput control and a centralized bandwidth control mechanism is also in-line with our work (Radhakrishnan et al., 2012).

III. BANDWIDTH CONTROL CONCEPT

This work rests on the concept of providing an assured minimum of bandwidth to the tenant VM's connections. Consider Figure III.1, which shows a bandwidth allocation of 1000Mbps to a guest virtual machine, for incoming traffic. Four bandwidth groups are requested by the VM and granted by the cloud bandwidth controller. The tenant VM has assigned one network connection per group. In this configuration, network connections one through four are assured a minimum bandwidth of 250 Mbps each of the total 1000 Mbps for incoming data traffic. There are two modes of operation as incoming traffic is consumed by each connection, normal mode and enforcement mode.

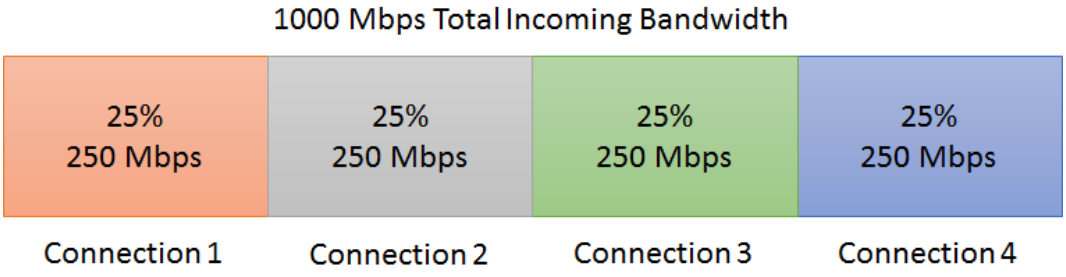


Figure III.1: Example Bandwidth Groups with Fixed Bandwidth Allocation

Unexceeded Bandwidth Utilization

In normal mode, the total bandwidth allocated is not exceeded. Hence the bandwidth groupings may exceed their assigned group bandwidth allocation, provided there is free available bandwidth. This allows for variability and flexibility for network connections within the groups to use the required bandwidth, provided the total bandwidth allotted is not exceeded. As shown in Figure III.2, during times T1 through T4 the bandwidth group utilization may grow beyond the group allocation provided there is enough available bandwidth to use for the time quanta (for our purposes, one second).

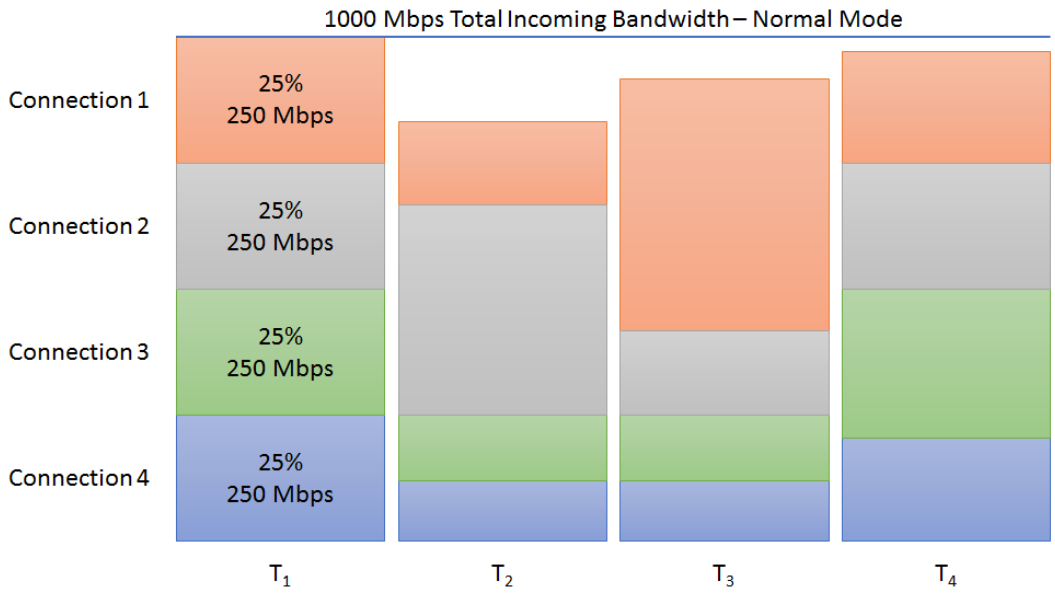


Figure III.2: Example Bandwidth Groups with Variable Utilization

Bandwidth Groups and the Free Pool

Each group has an actual limit, and a configured limit. As shown in Figure III.3, the bandwidth controller will analyze at each time quanta which groups have not exceeded their actual group bandwidth limit, providing the available bandwidth for other groups to grow. If a bandwidth group has not utilized all of it's configured bandwidth, then the actual group limit will be lowered by a (configurable) fixed amount (15000 bytes) at the end of each quanta. As groups shrink, their unused bandwidth is contributed to the "free pool", allowing groups that wish to grow to draw from the free pool.

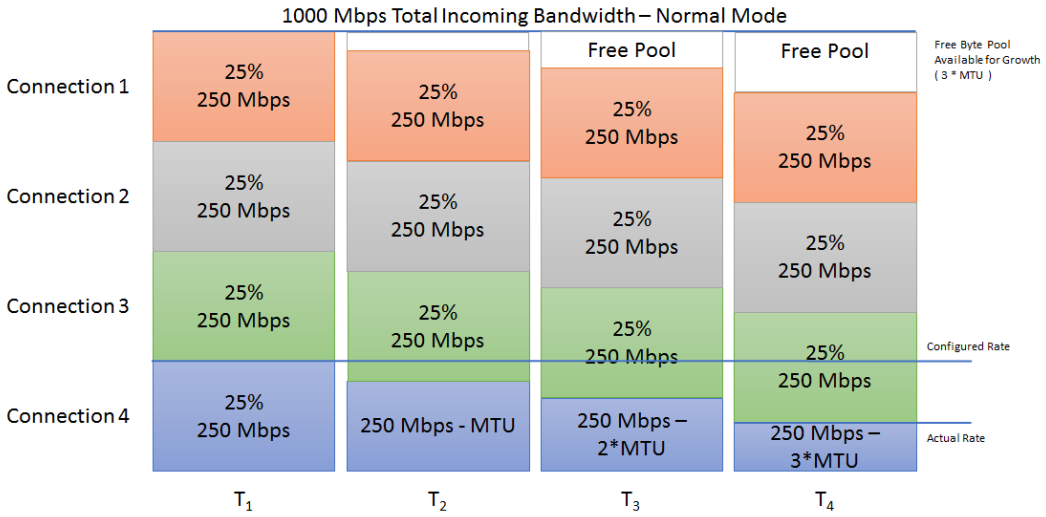


Figure III.3: Example Bandwidth Groups and the Free Pool

Bandwidth Group Depression

We see in Figure III.4 how a group grows beyond its configured limit, consuming available bandwidth from the free pool. This begs the question, what if a bandwidth group wishes to grow beyond its "actual rate" and take up bandwidth that was guaranteed by the "configured rate". In Figure III.4, we see connection four was 'depressed' from its configured rate of 25% to its actual rate of 10% solely because only ten percent was being utilized. Then connection one grew from its configured rate of 25% to its expanded rate of 40% of total allocated bandwidth, because free bandwidth was available due to the depression of the bandwidth group containing connection four. We consider now the case where connection four wishes to utilize its configured bandwidth.

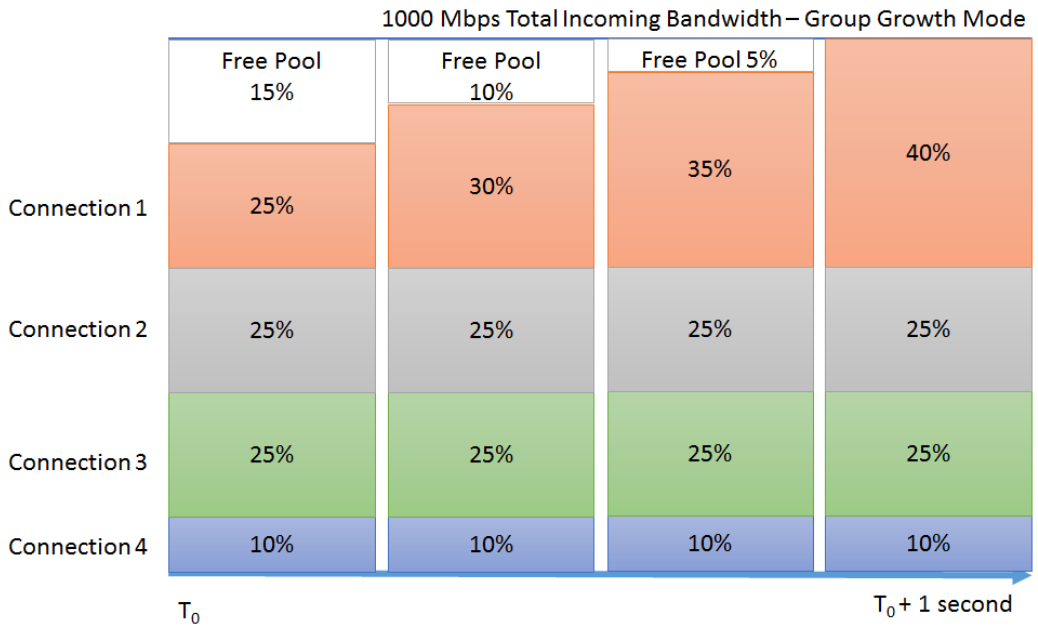


Figure III.4: Example Bandwidth Groups Growth

Resetting Bandwidth Group Limits

As shown in Figure III.5, connection four's actual limit was depressed to 10% of available bandwidth. Due to a demand of network bandwidth, the minimum configured guarantee of 25% to the bandwidth group for connection four must be satisfied. In addition, the bandwidth group containing connection one was only guaranteed a minimum of 25% even though it grew to use all remaining bandwidth (40% of maximum allowed). We therefore will reset all bandwidth group's actual bandwidth limit to equal their configured limit, satisfying both requirements.

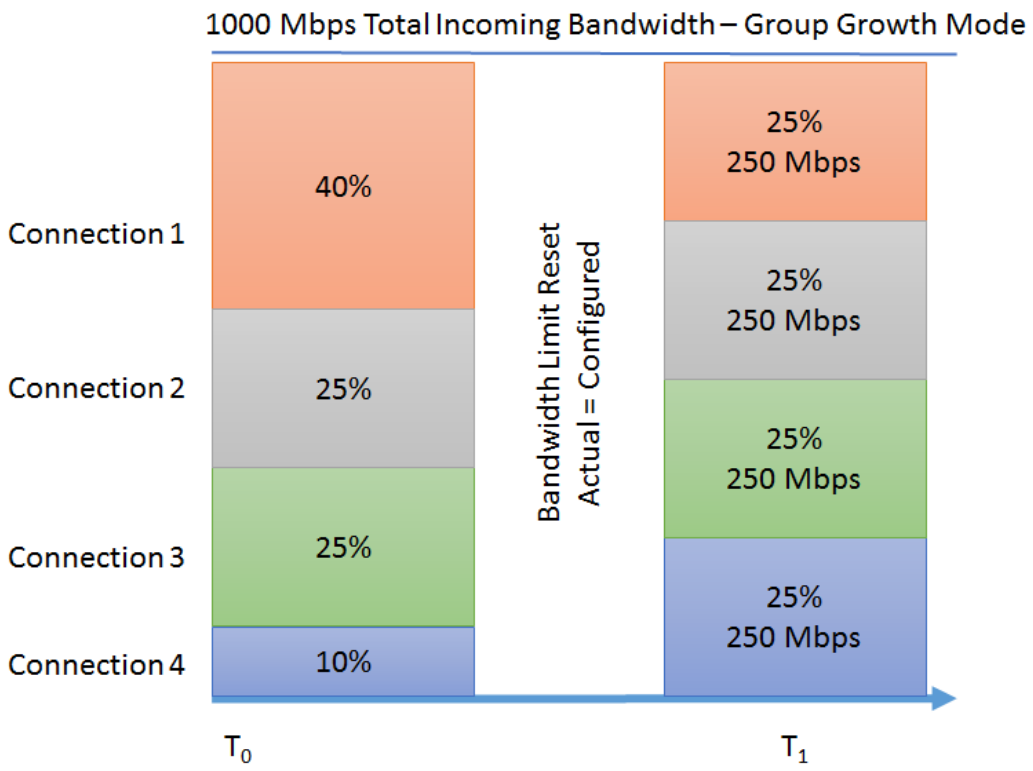


Figure III.5: Example Bandwidth Groups Limit Reset

Finally, consider the situation where the total bandwidth limitation was exceeded as shown in Figure III.6. Identifying the specific bandwidth group which exceeds the total bandwidth limitation is not necessary to the algorithm for the situation to be remedied, since the actual bandwidth limits are simply reset to their configured limits. In this case, reaching 110% of the bandwidth maximum limit is algorithmically impossible since packets would be dropped (and his represented at 10% for illustration purposes only). In the following quanta, the reset takes place. A bandwidth group enters "enforcement" mode when the algorithm drops packets to maintain bandwidth assurances.

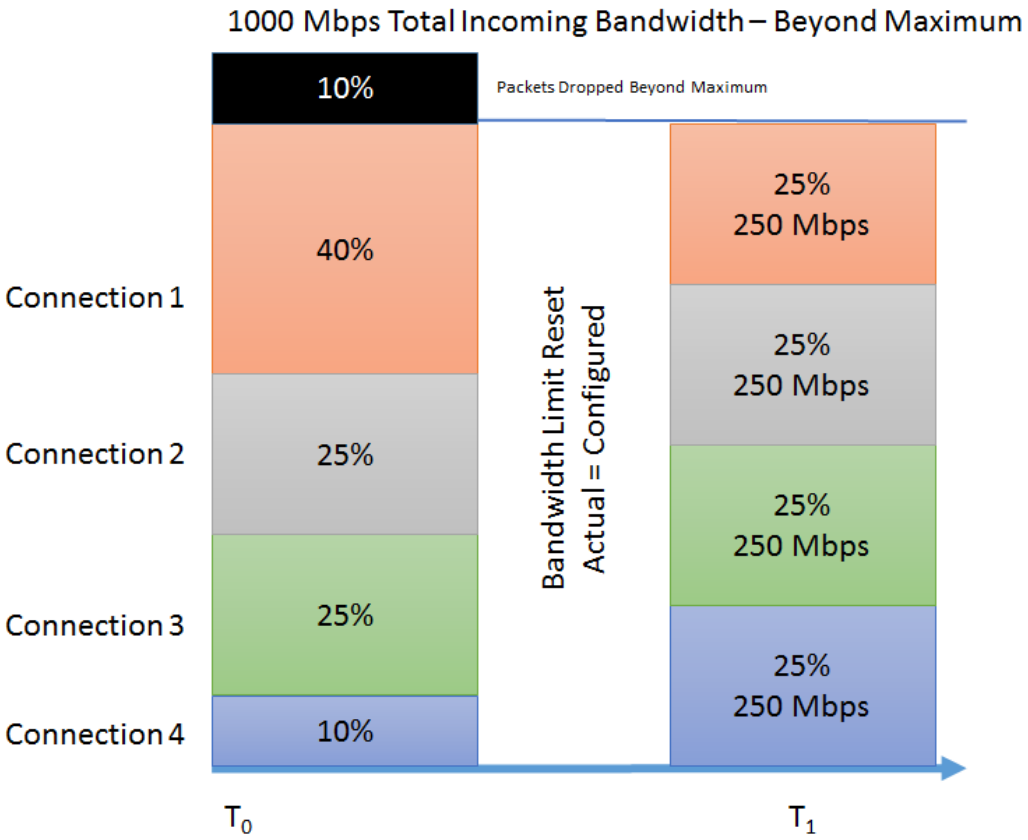


Figure III.6: Example Bandwidth Groups Exceed Maximum

IV. BANDWIDTH CONTROLLER ALGORITHM

Here we discuss the algorithm and data flow for the conceptual Cloud bandwidth controller. There are two events where the algorithm logic is triggered, when data packets are sent or received, and when the quanta timer fires. The logic is broken up into two directions, sending and receiving. Other than the direction of data flow, the logic is identical.

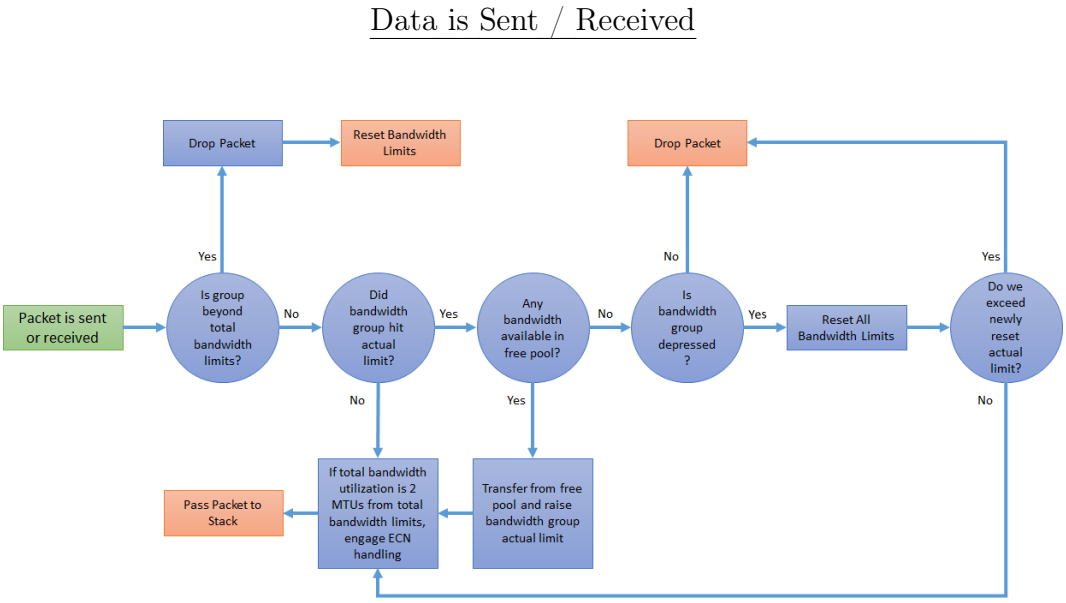


Figure IV.1: Algorithm Data Flow

In Figure IV.1, green represents the start of the algorithm, blue represents algorithm steps, and orange represents terminating events. This logic applies to packets that are sent and received on a network connection contained within a bandwidth group. Terminating events imply the algorithm logic has no further actions until another data packet is sent and received or the quanta timer fires. Resetting of bandwidth limits means that the actual bandwidth limits are reset to their configured limits, before bandwidth depression occurs due to lack of bandwidth utilization. If the data packet is within the bandwidth limitation, it is passed to the SCTP protocol stack for processing and forwarding to the application layer. If

the data packet is beyond the bandwidth limitation, it is dropped. This methodology has dual effects. First, the endpoint actively drops the packet to prevent processing packets beyond the maximum bandwidth limit. Secondly, the action of dropping the packet on a reliable transport connection (SCTP) is a mechanism to indirectly notify the sender that data loss has occurred, resulting in a retransmission (RTX). Since the packet is dropped before the transport layer has a chance to record the packet arrival, no acknowledgment will be sent, causing the sender to hold off sending any more data until the retransmission timeout (RTO) has expired.

Finally, if the total bandwidth utilized for the current quanta is within 2 MTU[1] of the total bandwidth allowed for this virtual machine, explicit congestion notification (ECN) is engaged. For incoming bandwidth traffic, the CE[2] bit is set by the bandwidth controller for incoming packets, causing the SCTP protocol stack to send an ECNE[3] chunk to the peer, thereby reducing incoming bandwidth from the sender. For outgoing bandwidth traffic, the SCTP protocol stack is informed that it received an ECNE chunk (even if it really did not), which triggers the reduction of the outgoing congestion window to slow bandwidth utilization (Ye et al., 2005). By utilizing these mechanisms we are implicitly able to limit the bandwidth utilized by the sender.

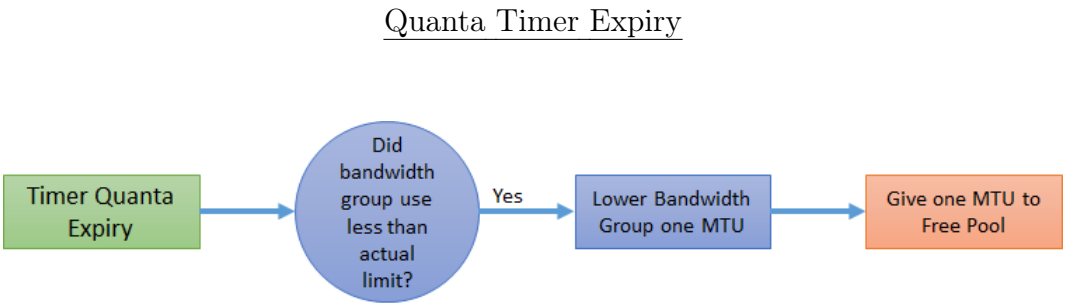


Figure IV.2: Quanta Timer Expiry

[1]packet maximum transmission unit size
[2]congestion encountered
[3]explicit congestion notification encountered

The algorithm depends on a quanta timer (default one second) upon which the bandwidth control mechanism is maintained. As shown in Figure IV.2, the timer expiry logic is executed for all configured bandwidth groups. This is the primary mechanism upon which the "free pool" is given bytes while borrowing bandwidth bytes from partitions that are not using it.

Software Implementation

The bandwidth controller algorithm is implemented as part of the SCTP kernel module within the Linux kernel. Requests to transmit data from the application are intercepted by the bandwidth controller algorithm. As shown in Figure IV.3, the decision is made whether to allow the packet to pass on to the transport layer (SCTP) and then to the Ethernet driver, or to be dropped due to bandwidth limitations. Incoming data follows a similar data flow where SCTP packets from the Ethernet driver are intercepted by the bandwidth controller algorithm before reaching the transport layer (SCTP). Here the decision is made whether to allow the packet to pass on to the transport layer (SCTP) and then to the application or to be dropped due to bandwidth limitations.

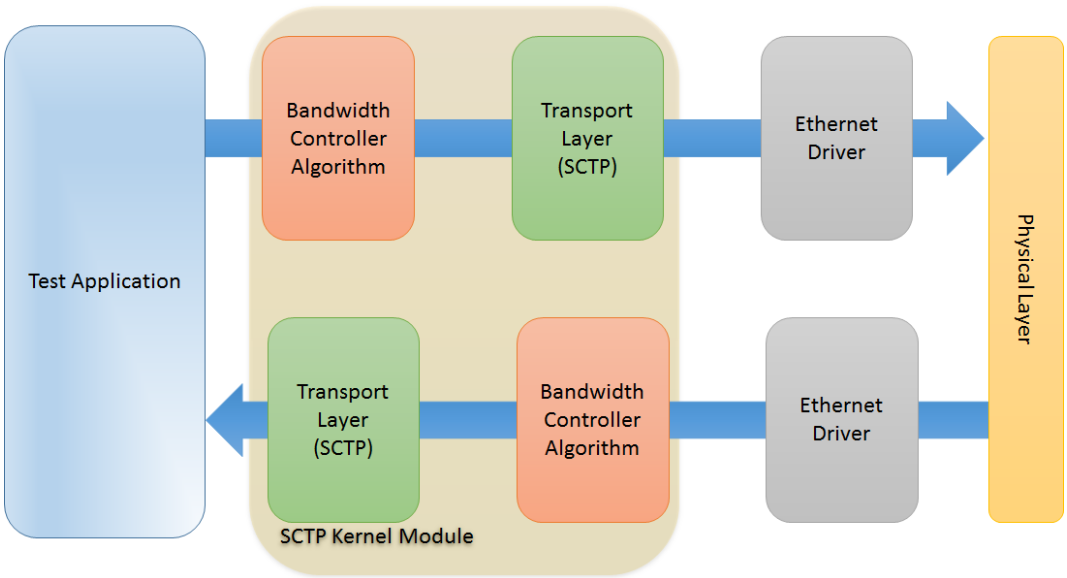


Figure IV.3: SCTP Bandwidth Controller

The advantage to this approach is that the transport layer (SCTP) is made

unaware of any actions taken by the bandwidth controller algorithm allowing the transport layer to commence reliable data operations (retransmissions, RTO, etc.) without changes to the SCTP transport algorithm. This also allows the bandwidth controller algorithm to drop packet data in a bidirectional fashion, relying on the guaranteed delivery nature of the transport layer to compensate, resulting in no loss of data.

V. EXPERIMENTAL DATA

The algorithm is experimentally verified by using virtual machines with four distinct connections between them, with each connection within it's own bandwidth group within both the server and client virtual machine. This data discusses the results of bandwidth limitation on incoming data from the point of view of a "server" which receives data from a client VM. All data discussed within this chapter is enumerated within the Appendix section of this thesis.

System Test Description

In order to simulate virtual computers communicating within a cloud environment, this research utilizes the Xen Project Hypervisor Server 4.4.1 running with Debian Linux 7.0 (Wheezy) x86-64 as dom0 (deb; xen). The Xen Server host hardware configuration consists of an Intel Core i5-3570K CPU @ 3.40 GHz with 8192 MB DDR3 RAM @ 1372 MHz. The Xen Server hosts two virtual machines, both running at default hypervisor configurations and identical software configurations. The only differences are the test roles. The guest virtual machines are running Arch Linux 2014.09.03 with Linux Kernel 3.16.3 using default kernel build parameters for Arch Linux (arc; ker). Each virtual machine is a uniprocessor with 512 MB of RAM allocated and runs within the Xen Hypervisor as a HVM (Hardware Virtual Machine) client (xen).

Test Description - Client Sends Data to Server with Bandwidth Limiting

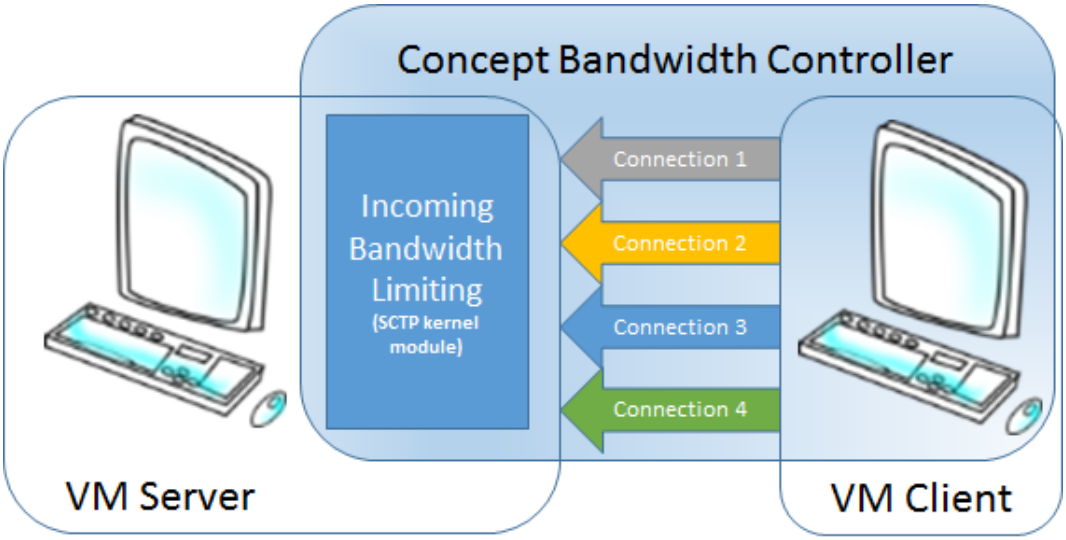


Figure V.1: Incoming Bandwidth Limiting Test

As shown in Figure V.1, the client VM has four processes with one SCTP connection each, where each register their connection with the "bandwidth controller" functionality added to the SCTP kernel module. Since these are separate SCTP connections between the client and the server, each connection will be subject to its own congestion window (Stewart and Xie, 2002). In this fashion, if one connection is bandwidth limited, the other connections are not affected. The client sends data for each test case in the following manner, with each step separated by a time delay of 30 seconds except where noted.

- Send no data on any connection (0 bytes/sec)
- All four connections send 10000 bytes/sec
- Connection 0 bandwidth increases by 5000 bytes/sec every thirty seconds until 30000 bytes/sec is reached
- Connection 1 bandwidth increases by 5000 bytes/sec every thirty seconds until 30000 bytes/sec is reached

- Connection 2 bandwidth increases by 5000 bytes/sec every thirty seconds until 30000 bytes/sec is reached
- Connection 3 bandwidth increases by 5000 bytes/sec every thirty seconds until 30000 bytes/sec is reached
- At this point the cumulative bandwidth sent to the server by the client VM is 120000 bytes/sec; the client continues for 300 seconds in this state
- Send no data on any connection (0 bytes/sec) - end of test

Each connection is contained within a bandwidth group on the server. The server virtual machine has a maximum bandwidth of 100000 bytes (for easy verification), and the group bandwidth allocations are 10% - 10000 bytes per second, 20% - 20000 bytes per second, 30% - 30000 bytes per second, and 40% - 40000 bytes per second on the server.

Server Receiving Data - Server Incoming Bandwidth Limited

The data reported in this section reflects bandwidth utilization from the point of view of the server VM. There is no bandwidth limitation on the client for this test case so all retransmissions and bandwidth limitations are an effect of the bandwidth limitation actions of the server.

Group 0 - 10% Bandwidth Group - 10000 bytes/sec Minimum Assured

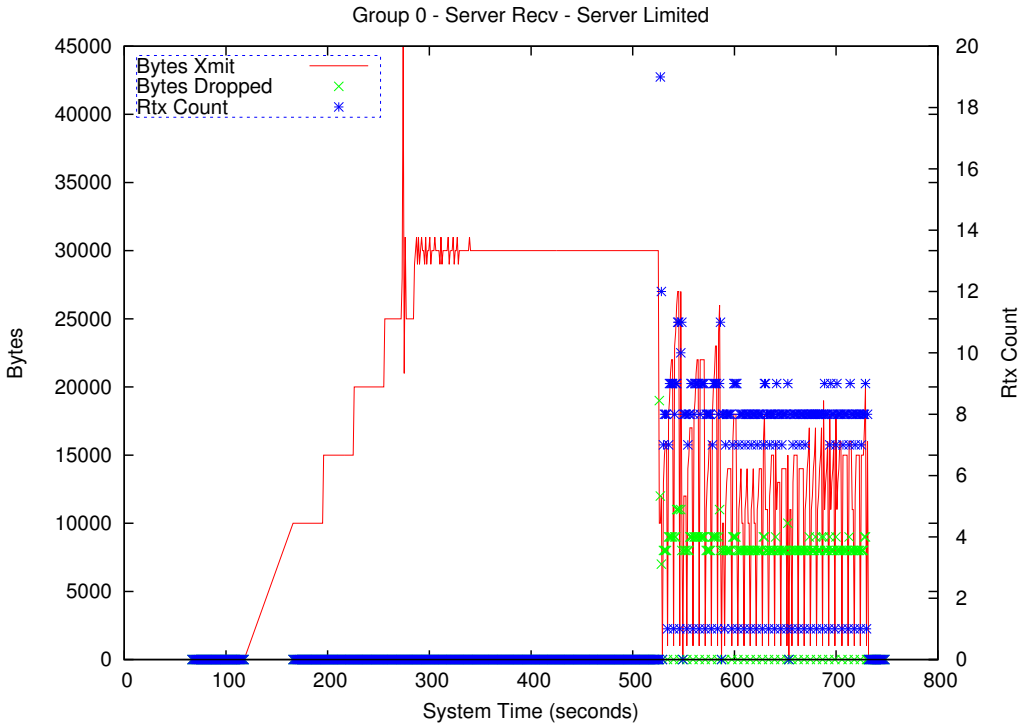


Figure V.2: Incoming Bandwidth Group 0 - 10000 bytes/sec Minimum Assured

The graph in Figure V.2 represents bytes sent every second for the 10% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. The assured bandwidth for this group is 10,000 bytes per second. At the 527 second mark, the total incoming bandwidth limit is reached, so the bandwidth for this group is now limited by packet drops to its configured 10% minimum assured bandwidth - 10,000 bytes per second. The connection within this bandwidth group is attempting to send more data after each retransmission and is being limited by bandwidth enforcement of the "bandwidth controller". As a result, the recorded bandwidth measurement shows a "zig-zag" pattern due to dropping incoming data packets by the bandwidth enforcement mechanism.

When packets are dropped, the client sender stops sending data to allow for the packet retransmission timeout (RTO) to expire before sending more data. These retransmissions are represented by the blue dots overlaid in Figure V.2,

and in subsequent figures. Each corresponding drop in bandwidth is associated with a retransmission packet shortly thereafter. The amount of data dropped is represented by the green dots overlaid in Figure V.2. This shows the amount of data that the client is attempting to send but was rejected by the bandwidth controller. Note that the period of time before we hit the total incoming bandwidth limit contains no retransmissions or dropped packets.

Group 1 - 20% Bandwidth Group - 20000 bytes/sec Minimum Assured

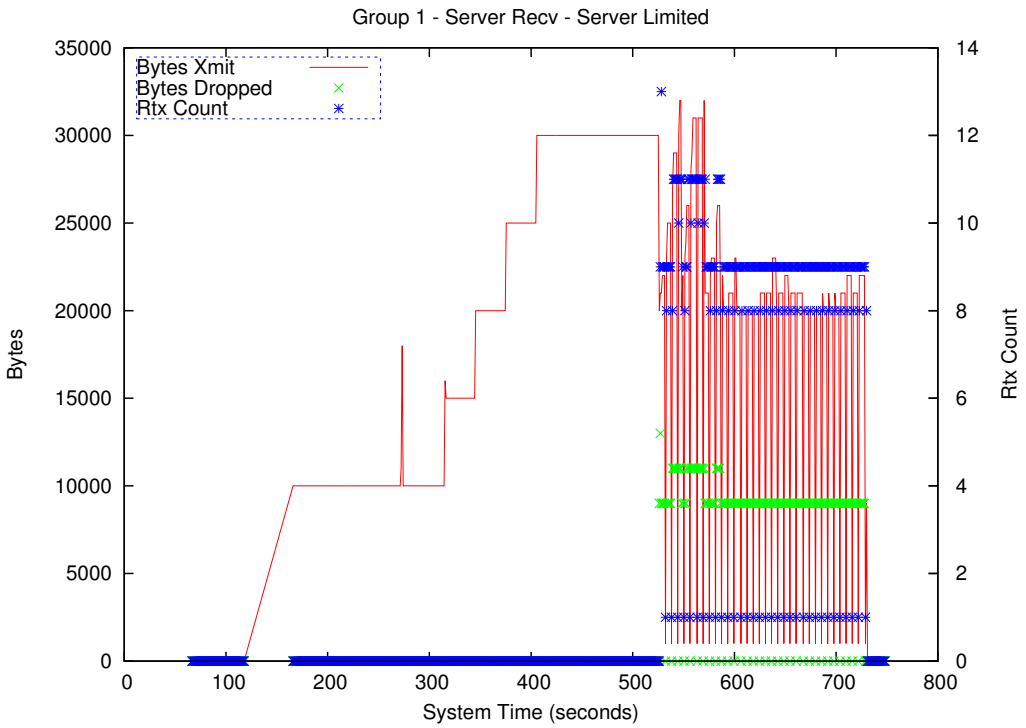


Figure V.3: Incoming Bandwidth Group 1 - 20000 bytes/sec Minimum Assured

The graph in Figure V.3 represents bytes sent every second for the 20% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. At the 527 second mark, the total incoming bandwidth limit for the server is reached, so the bandwidth for this group is now limited by packet drops to its configured 20% minimum assured bandwidth - 20,000 bytes per second. Random ordering of bandwidth utilization by connections between groups cause small spikes in bandwidth consumption. These

spikes still do not violate the total maximum bandwidth limitations and represent the "first come first serve" nature of bandwidth utilization between connections, utilizing what little bandwidth is left after the other bandwidth groups are granted their minimum assured allotment.

Group 2 - 30% Bandwidth Group - 30000 bytes/sec Minimum Assured

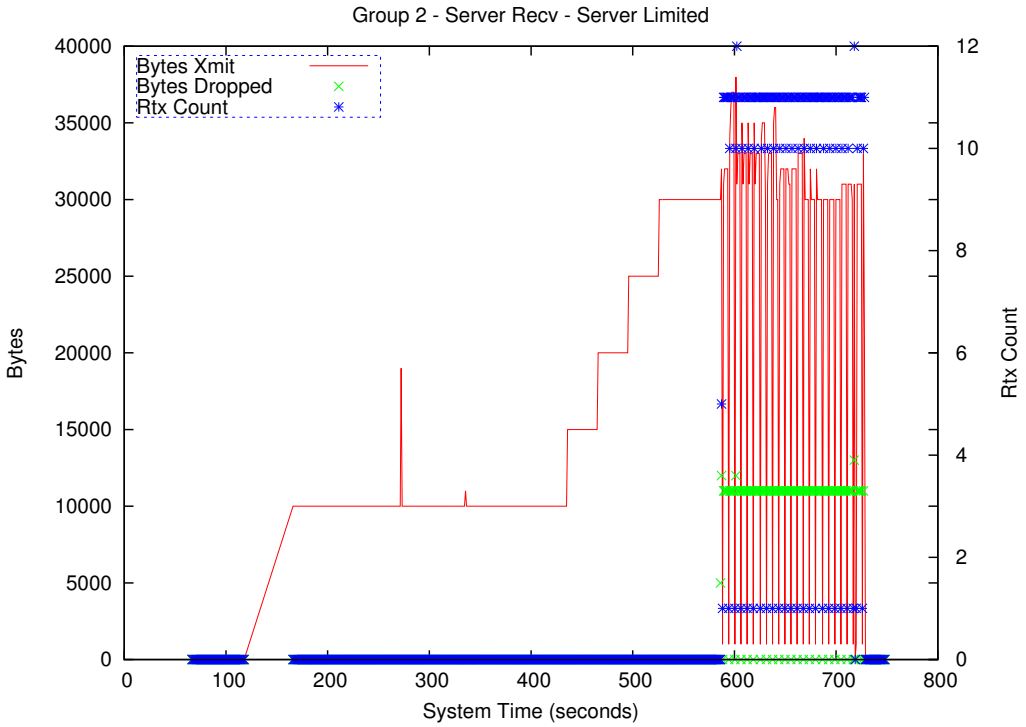


Figure V.4: Incoming Bandwidth Group 2 - 30000 bytes/sec Minimum Assured

The graph in Figure V.4 represents bytes sent every second for the 30% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. The assured bandwidth for this group is also 30,000 bytes per second. Since bandwidth utilization is very proximate to the minimum guaranteed bandwidth for this group, only one byte of data over the bandwidth limit will cause enforcement action on this group. For some time the group experiences no packet drops when this group hits the 30,000 byte per second test case, until the 585 second mark where transient overshoot of bandwidth utilization causes enforcement action to occur. To allow the

30% group the assured minimum bandwidth, the groups that are exceeding their assured minimum bandwidth must experience packet drops to limit excess bandwidth utilization. Additionally, this group must also be enforced to ensure other groups receive their alloted minimum bandwidth guarantee.

Group 3 - 40% Bandwidth Group - 40000 bytes/sec Minimum Assured

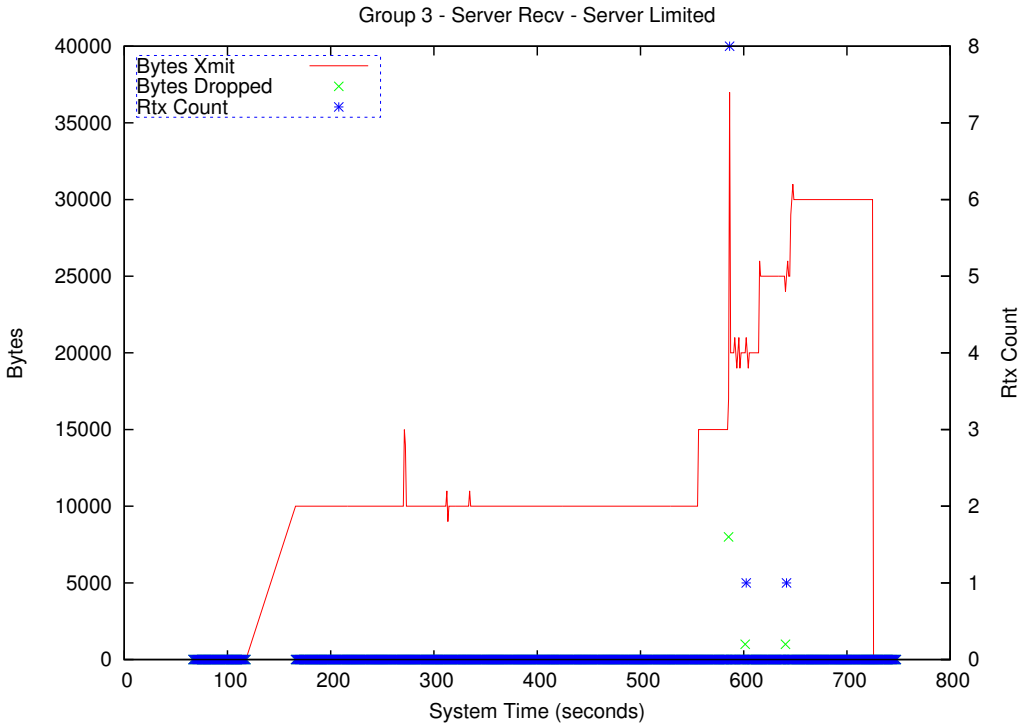


Figure V.5: Incoming Bandwidth Group 3 - 40000 bytes/sec Minimum Assured

The graph in Figure V.5 represents bytes sent every second for the 40% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. The server has assured this connection group a bandwidth of 40,000 bytes per second, of which 30,000 bytes per second is utilized. Therefore no packet retransmissions are observed with the exception of a handful of retransmission events recorded at 586, 602 and 641 seconds; see the section entitled "Issues and Concerns" regarding "Covetous Bandwidth Effect".

Since the assured minimum is 40,000 bytes per second but only 30,000 bytes

per second are utilized, 10,000 bytes per second of bandwidth are available for other bandwidth groups to consume. This allows other groups (like the 10% and 20% bandwidth groups shown in figures V.2 and V.3, respectively) to attempt to grow beyond their configured limit while still experiencing bandwidth drops. In this fashion the maximum bandwidth allowed is always respected while continuing to provide as much bandwidth available to the other groups.

Client Sending Data - Server Incoming Bandwidth Limited

The data reported in this section reflects bandwidth utilization from the point of view of the client VM. There is no bandwidth limitation on the client for this test case so all retransmissions and bandwidth limitations are an effect of the bandwidth limitation actions of the server.

Group 0 - 10% Bandwidth Group - 10000 bytes/sec Minimum Assured

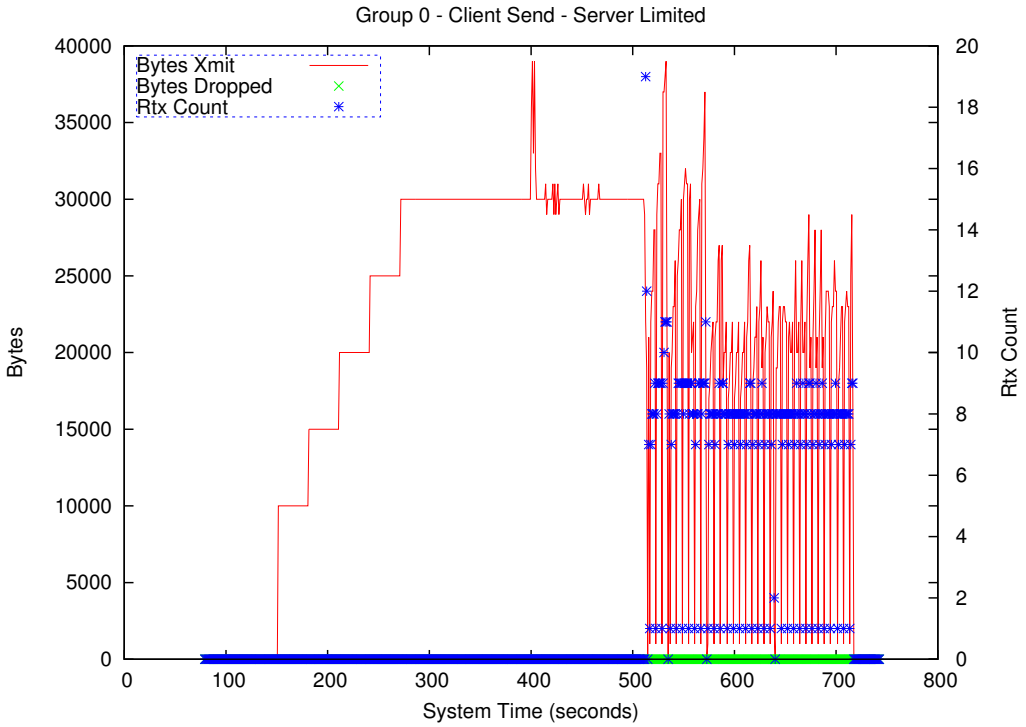


Figure V.6: Outgoing Bandwidth Group 0 - 10000 bytes/sec Minimum Assured

The graph in Figure V.6 represents bytes sent every second for the 10% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. At the 512 second mark, the total incoming bandwidth limit for the server is reached, so the bandwidth for this group is now limited with packet drops by the server, since this group was only configured for 10000 bytes per second minimum. Notice the increase in retransmissions after this point in time. This demonstrates an effect of the client attempting to send more than is allowed by the server. It is interesting that the SCTP transport protocol did not adjust the congestion window size accordingly; this is discussed in the section VII titled "Issues and Concerns".

Group 1 - 20% Bandwidth Group - 20000 bytes/sec Minimum Assured

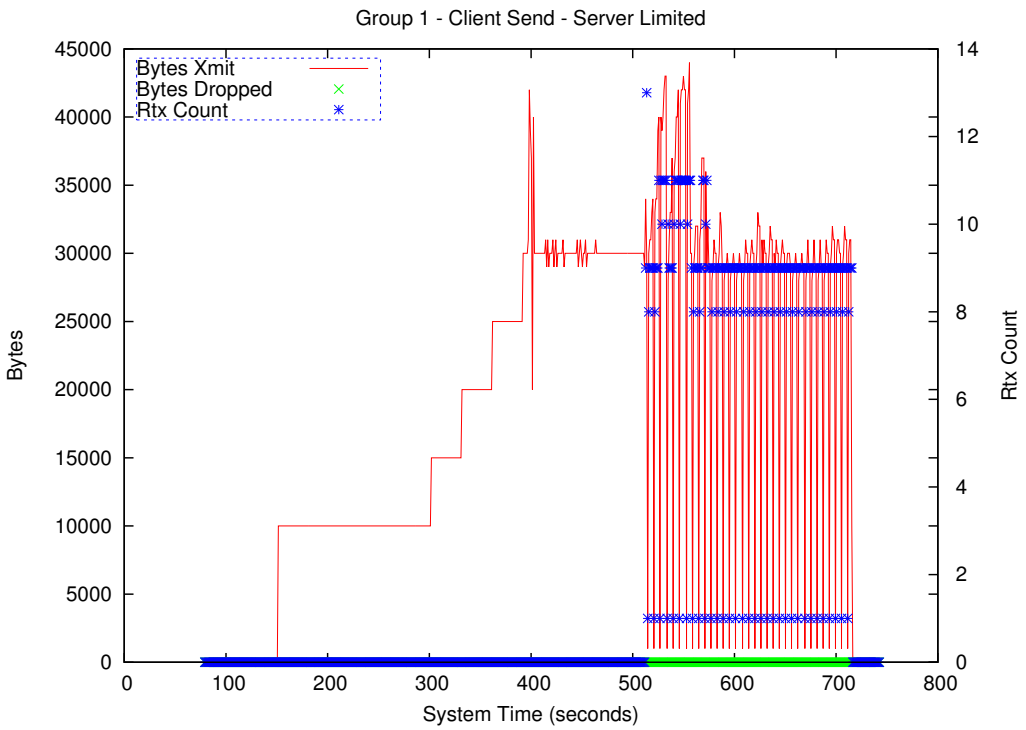


Figure V.7: Outgoing Bandwidth Group 1 - 20000 bytes/sec Minimum Assured

The graph in Figure V.7 represents bytes sent every second for the 20% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. At the 512 second mark,

the total incoming bandwidth limit for the server is reached, so the bandwidth for this group is now limited with packet drops by the server, since this group was only configured for 20000 bytes per second minimum. Notice the increase in retransmissions after this point in time. This demonstrates an effect of the client attempting to send more than is allowed by the server. It is interesting that the SCTP transport protocol did not adjust the congestion window size accordingly; this is discussed in the section VII titled "Issues and Concerns".

Group 2 - 30% Bandwidth Group - 30000 bytes/sec Minimum Assured

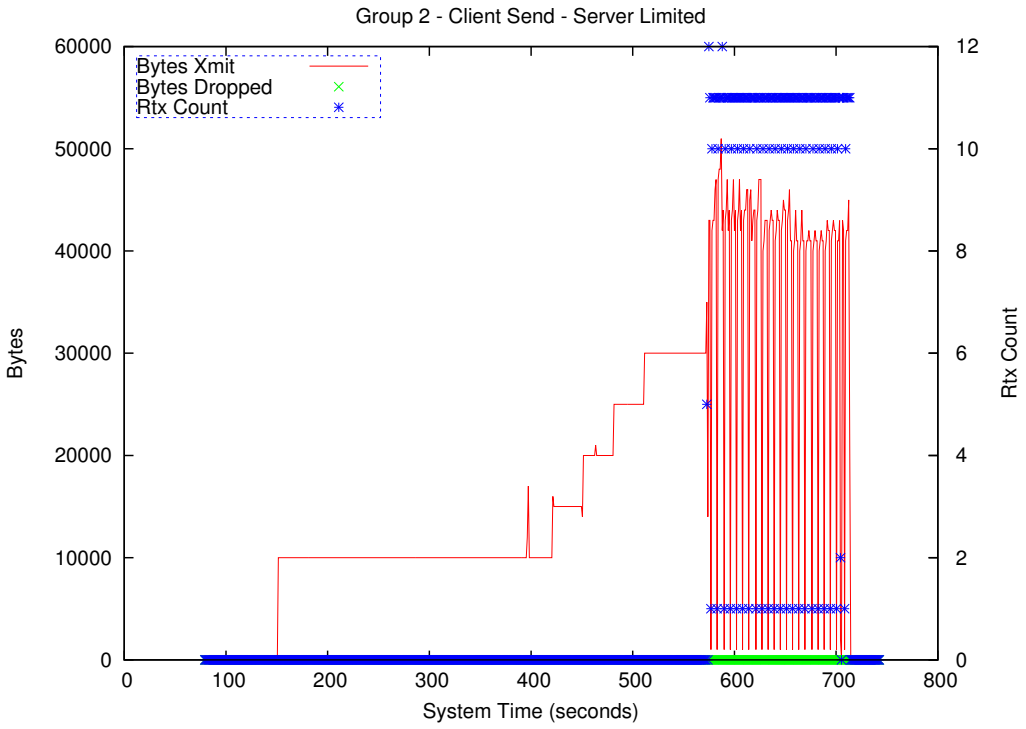


Figure V.8: Outgoing Bandwidth Group 2 - 30000 bytes/sec Minimum Assured

The graph in Figure V.8 represents bytes sent every second for the 30% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. The assured bandwidth for this group is also 30,000 bytes per second. Similar to Figure V.4, retransmissions are present for this group. For some time the group experiences no packet drops when this group hits the 30,000 byte per second test case, until the 572 second mark

where transient overshoot of bandwidth utilization causes enforcement action to occur. To allow the 30% group the assured minimum bandwidth, the groups that are exceeding their assured minimum bandwidth must experience packet drops to limit excess bandwidth utilization. Additionally, this group must also be enforced to ensure other groups receive their alloted minimum bandwidth guarantee.

Group 3 - 40% Bandwidth Group - 40000 bytes/sec Minimum Assured

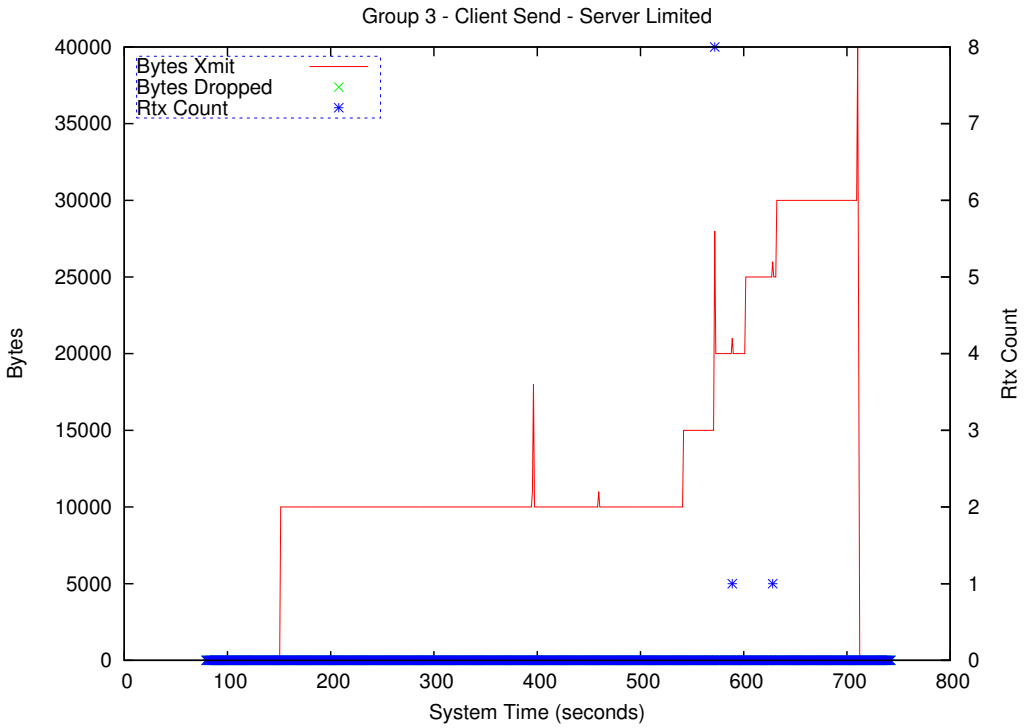


Figure V.9: Outgoing Bandwidth Group 3 - 40000 bytes/sec Minimum Assured

The graph in Figure V.9 represents bytes sent every second for the 40% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. The server has assured this connection group a bandwidth of 40,000 bytes per second, of which 30,000 bytes per second is utilized. Therefore no packet retransmissions are observed with the exception of a handful of retransmission events recorded at 571, 588, and 628 seconds; see the section entitled "Issues and Concerns" regarding "Covetous Bandwidth Effect".

Since the assured minimum is 40,000 bytes per second but only 30,000 bytes per second are utilized, 10,000 bytes per second of bandwidth are available for other bandwidth groups to consume. This allows other groups (like the 10% and 20% bandwidth groups shown in figures V.6 and V.7, respectively) to attempt to grow beyond their configured limit while still experiencing bandwidth drops. In this fashion the maximum bandwidth allowed is always respected while continuing to provide as much bandwidth available to the other groups.

Test Description - Bandwidth Limited Client Sends Data to Server

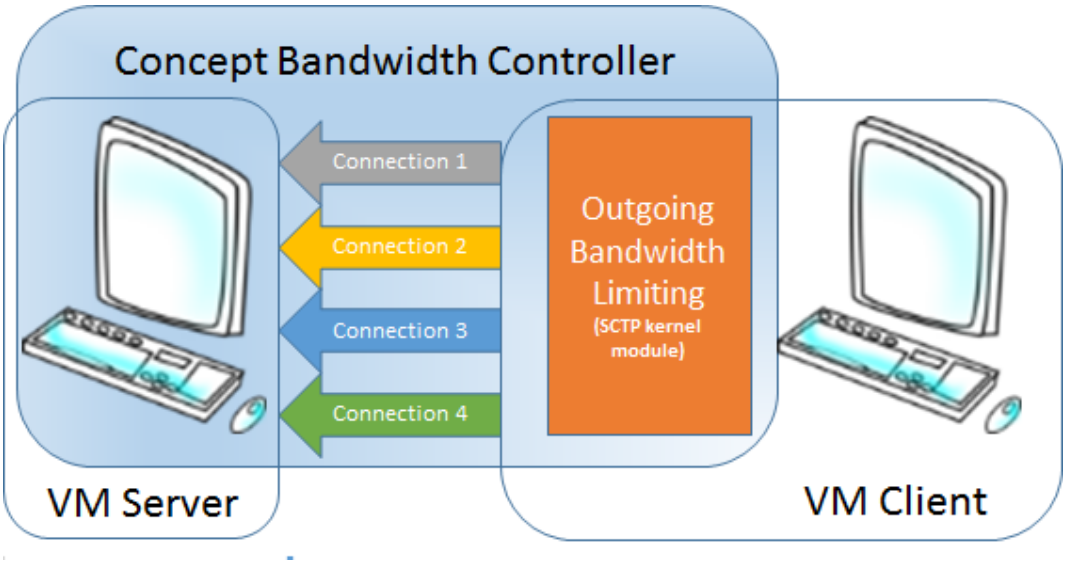


Figure V.10: Outgoing Bandwidth Limiting Test

In this test, bandwidth limiting occurs on the outgoing bandwidth of the client VM; the receiving server VM bandwidth is unlimited. All other test parameters are identical including bandwidth group allocations and bandwidth utilization.

Client Sending Data - Client Outgoing Bandwidth Limited

The data reported in this section reflects bandwidth utilization from the point of view of the client VM. There is no bandwidth limitation on the server for this test case so all retransmissions and bandwidth limitations are an effect of the bandwidth limitation actions of the client.

Group 0 - 10% Bandwidth Group - 10000 bytes/sec Minimum Assured

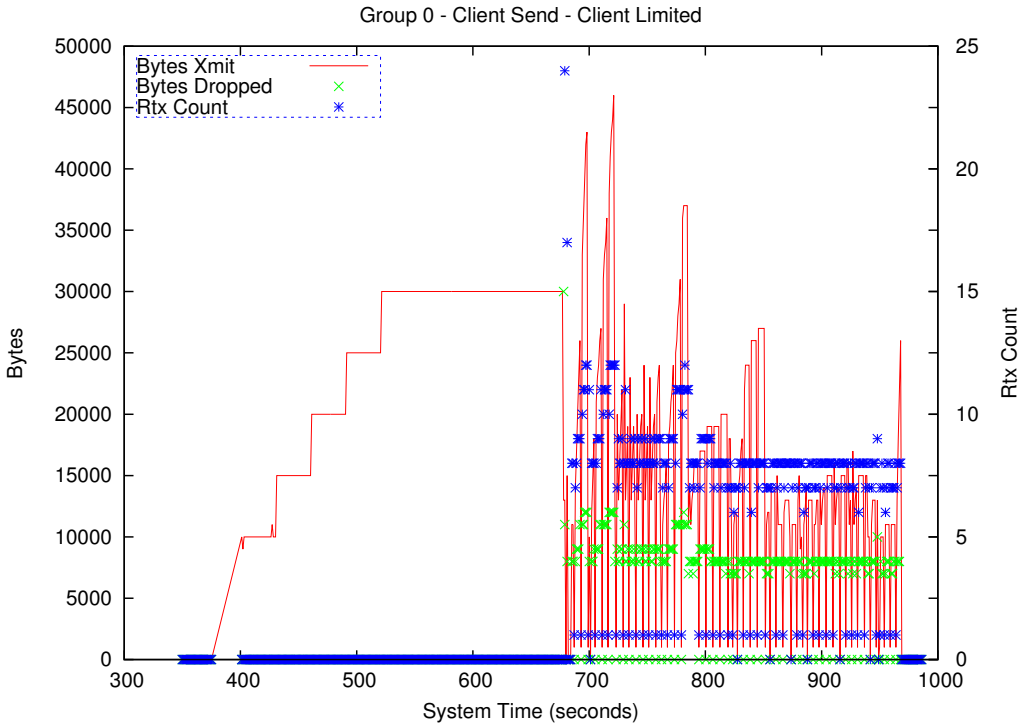


Figure V.11: Outgoing Bandwidth Group 0 - 10000 bytes/sec Minimum Assured

The graph in Figure V.11 represents bytes sent every second for the 10% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. At the 679 second mark, the total outgoing bandwidth limit for the client is reached, so the bandwidth for this group is now limited with packet drops by the client, since this group was only configured for 10000 bytes per second minimum. Notice the increase in retransmissions after this point in time. This demonstrates an effect of the client attempting to send more than is allowed by the server. It is interesting that the SCTP transport protocol did not adjust the congestion window size accordingly; this is discussed in the section VII titled "Issues and Concerns".

Group 1 - 20% Bandwidth Group - 20000 bytes/sec Minimum Assured

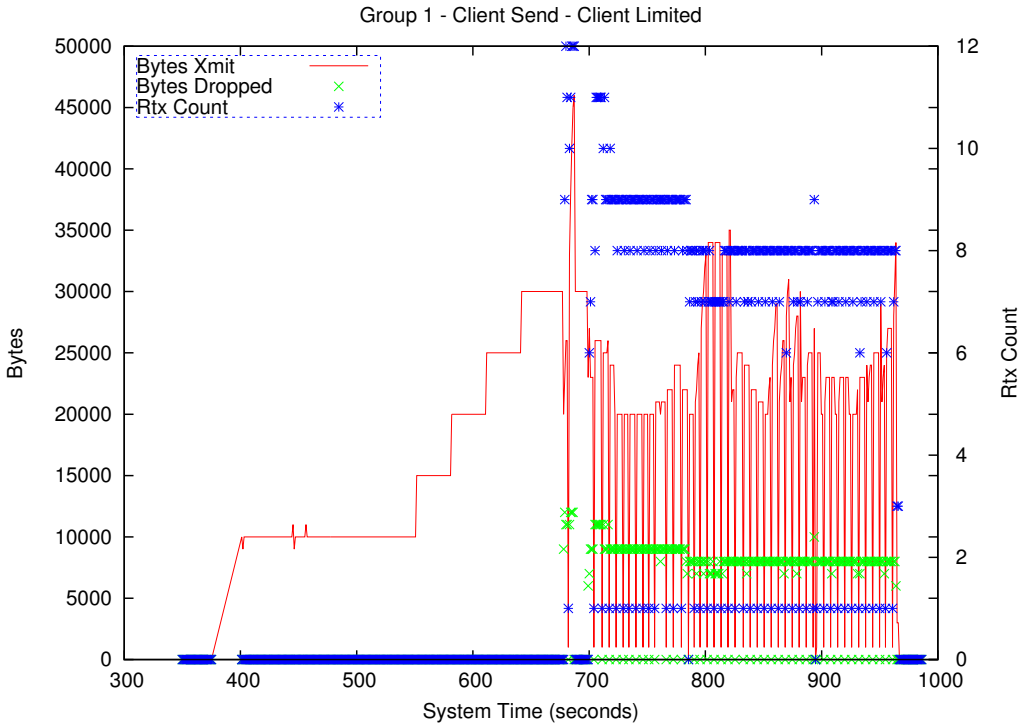


Figure V.12: Outgoing Bandwidth Group 1 - 20000 bytes/sec Minimum Assured

The graph in Figure V.12 represents bytes sent every second for the 20% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. At the 679 second mark, the total outgoing bandwidth limit for the client is reached, so the bandwidth for this group is now limited with packet drops by the client, since this group was only configured for 20000 bytes per second minimum. Notice the increase in retransmissions after this point in time. This demonstrates an effect of the client attempting to send more than is allowed by the server. It is interesting that the SCTP transport protocol did not adjust the congestion window size accordingly; this is discussed in the section VII titled "Issues and Concerns".

Group 2 - 30% Bandwidth Group - 30000 bytes/sec Minimum Assured

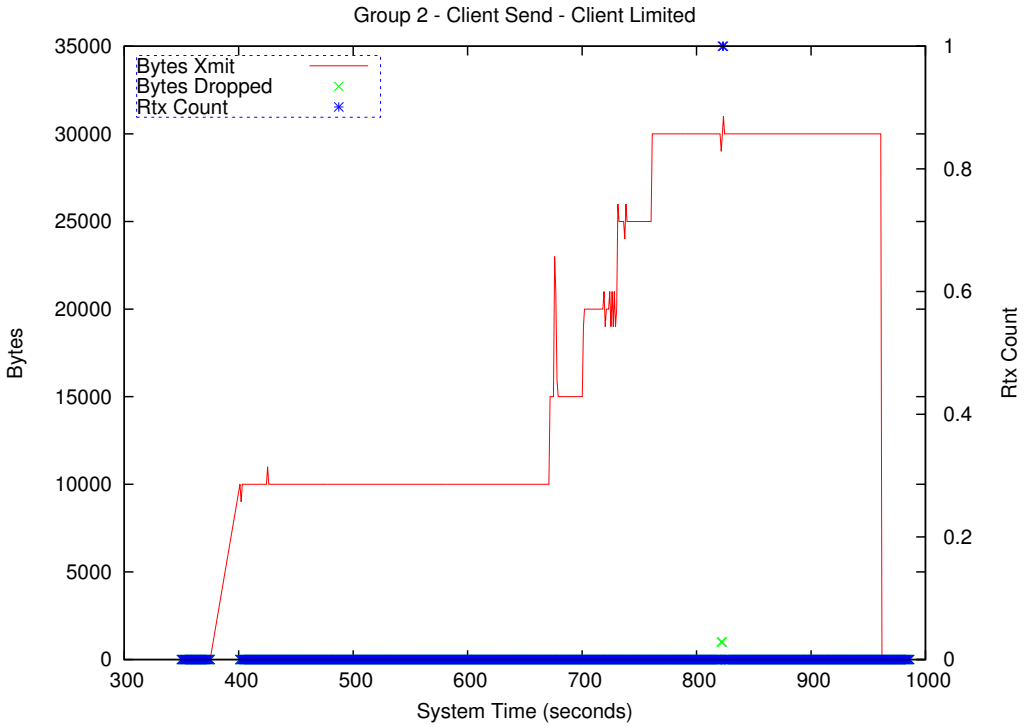


Figure V.13: Outgoing Bandwidth Group 2 - 30000 bytes/sec Minimum Assured

The graph in Figure V.13 represents bytes sent every second for the 30% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. The client has assured this connection group a bandwidth of 30,000 bytes per second, which is fully utilized by the client. Therefore no packet retransmissions are observed with the exception of two retransmission events at 822 and 823 seconds; see the section titled "Issues and Concerns" regarding "Covetous Bandwidth Effect".

Group 3 - 40% Bandwidth Group - 40000 bytes/sec Minimum Assured

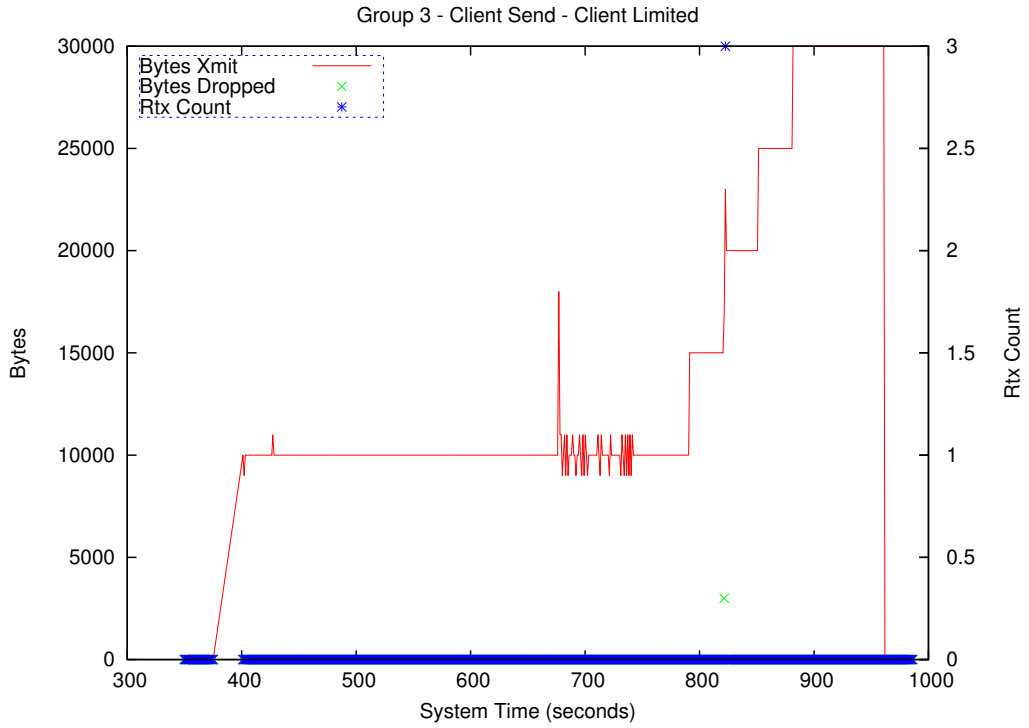


Figure V.14: Outgoing Bandwidth Group 3 - 40000 bytes/sec Minimum Assured

The graph in Figure V.14 represents bytes sent every second for the 40% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. The client has assured this connection group a bandwidth of 40,000 bytes per second, of which 30,000 bytes per second is utilized. Therefore no packet retransmissions are observed but three, recorded at 822 seconds; see the section titled "Issues and Concerns" regarding "Covetous Bandwidth Effect".

Since the assured minimum is 40,000 bytes per second but only 30,000 bytes per second are utilized, 10,000 bytes per second of bandwidth are available for other bandwidth groups to consume. This allows other groups (like the 10% and 20% bandwidth groups shown in figures V.11 and V.12, respectively) to attempt to grow beyond their configured limit while still experiencing bandwidth drops. In this fashion the maximum bandwidth allowed is always respected while continuing to provide as much bandwidth available to the other groups.

Server Receiving Data - Client Outgoing Bandwidth Limited

The data reported in this section reflects bandwidth utilization from the point of view of the server VM. There is no bandwidth limitation on the server for this test case so all retransmissions and bandwidth limitations are an effect of the bandwidth limitation actions of the client.

Group 0 - 10% Bandwidth Group - 10000 bytes/sec Minimum Assured

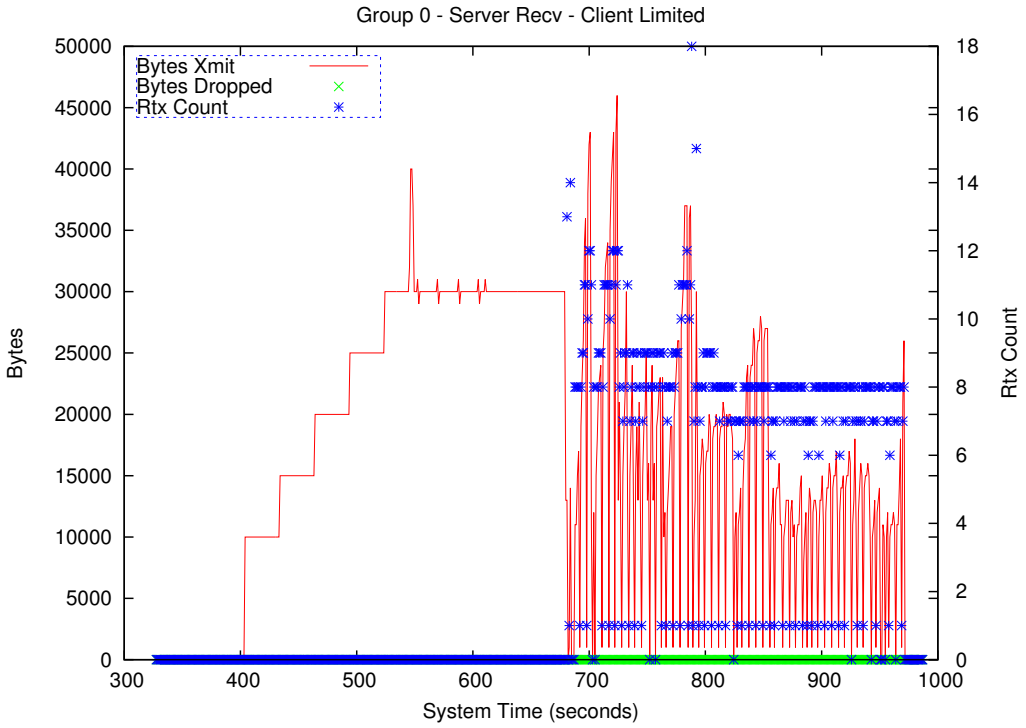


Figure V.15: Incoming Bandwidth Group 0 - 10000 bytes/sec Minimum Assured

The graph in Figure V.15 represents bytes sent every second for the 10% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. At the 680 second mark, the total outgoing bandwidth limit is reached by the client so the bandwidth for this group is now limited by packet drops, since this group was only configured for 10000 bytes per second minimum. Notice the increase in retransmissions after this point in time. This demonstrates an effect of the client attempting to send more than is allowed by the server. It is interesting that the SCTP transport protocol

did not adjust the congestion window size accordingly; this is discussed in the section VII titled "Issues and Concerns".

Group 1 - 20% Bandwidth Group - 20000 bytes/sec Minimum Assured

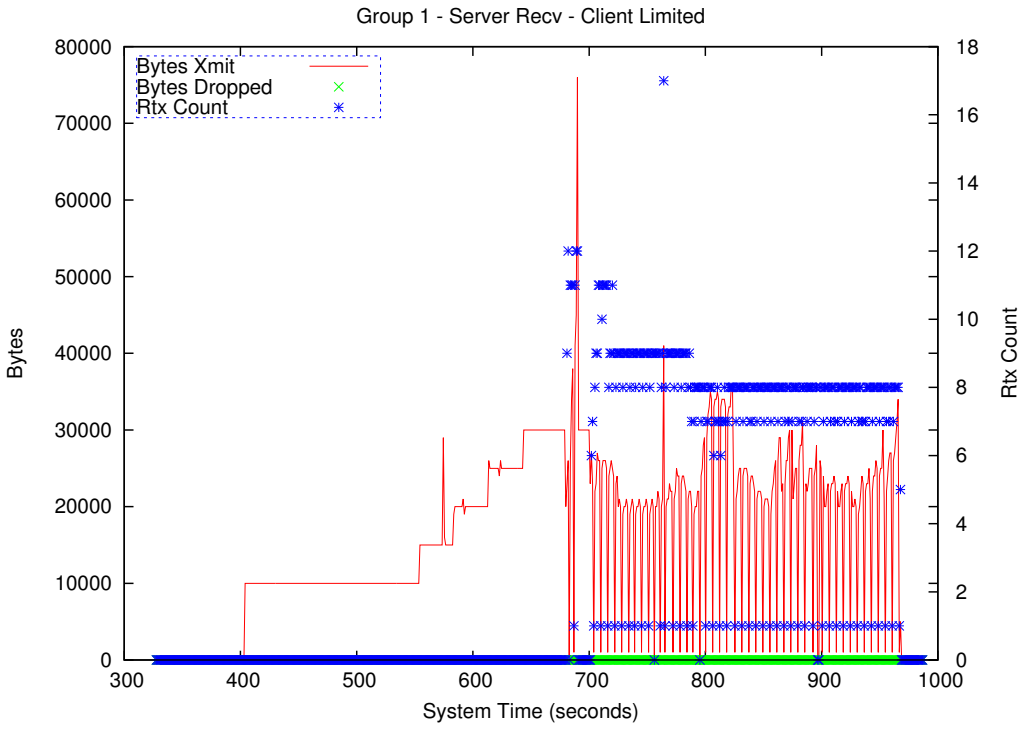


Figure V.16: Incoming Bandwidth Group 1 - 20000 bytes/sec Minimum Assured

The graph in Figure V.16 represents bytes sent every second for the 20% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. At the 680 second mark, the total outgoing bandwidth limit is reached by the client, so the bandwidth for this group is now limited by packet drops, since this group was only configured for 20000 bytes per second minimum. Notice the increase in retransmissions after this point in time. This demonstrates an effect of the client attempting to send more than is allowed by the server. It is interesting that the SCTP transport protocol did not adjust the congestion window size accordingly; this is discussed in the section VII titled "Issues and Concerns".

Group 2 - 30% Bandwidth Group - 30000 bytes/sec Minimum Assured

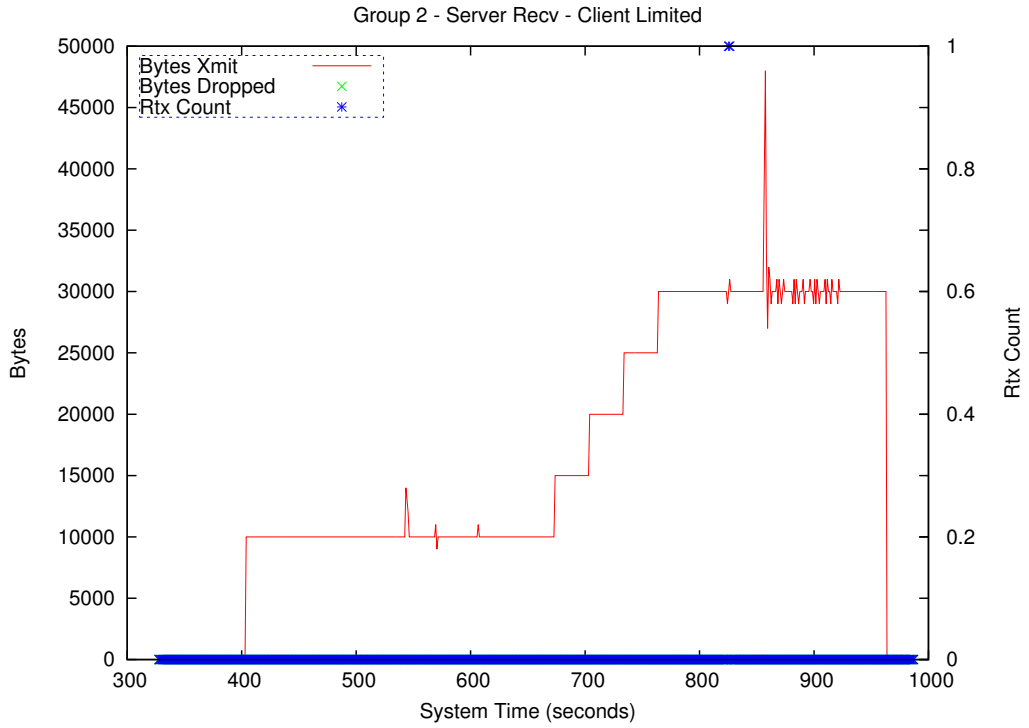


Figure V.17: Incoming Bandwidth Group 2 - 30000 bytes/sec Minimum Assured

The graph in Figure V.17 represents bytes sent every second for the 30% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. The client has assured and utilizes this connection group a bandwidth of 30,000 bytes per second. Therefore, no packet retransmissions are observed from the server perspective with the exception of two retransmission events at 825 and 826 seconds. We call this the "covetous bandwidth effect", described in the "Issues and Concerns" section.

Group 3 - 40% Bandwidth Group - 40000 bytes/sec Minimum Assured

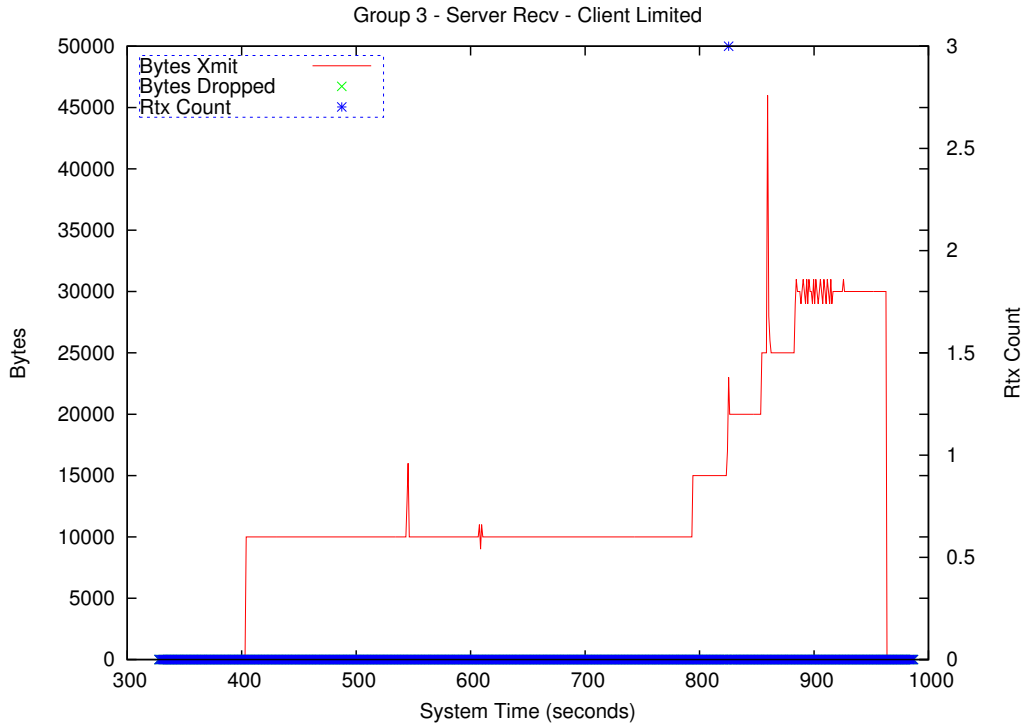


Figure V.18: Incoming Bandwidth Group 3 - 40000 bytes/sec Minimum Assured

The graph in Figure V.18 represents bytes sent every second for the 40% bandwidth group. Over time, the connection within the group utilizes more and more bandwidth, holding steady at 30,000 bytes per second. Since the actual bandwidth utilized is less than the assured minimum bandwidth, no enforcement mechanism is employed against this bandwidth group with the exception of a handful of retransmission events recorded at 825 seconds. We call this the "covetous bandwidth effect", described in the "Issues and Concerns" section.

Since the assured minimum is 40,000 bytes per second but only 30,000 bytes per second are utilized, 10,000 bytes per second of bandwidth are available for other bandwidth groups to consume. This allows other groups (like the 10% and 20% bandwidth groups shown in figures V.15 and V.16, respectively) to attempt to grow beyond their configured limit while still experiencing bandwidth drops. In this fashion the maximum bandwidth allowed is always respected while continuing to provide as much bandwidth available to the other groups.

Comparison with Uncategorized Bandwidth Control

While the results of our new concept of bandwidth control are illuminating, it is helpful to compare our work against the most basic form of bandwidth control; raw (uncategorized) bandwidth limiting at the network interface level. The Xen hypervisor comes with a traffic control mechanism which makes use of the Linux Kernel to provide network bandwidth limitations. This is accomplished by imposing queuing discipline on virtual network interfaces for each virtual machine hosted by the hypervisor. Queuing discipline allows for packets intended for a virtualized network device (interface) are appended into a queue for transmission. If the queue is full, then packets are dropped. The size and type of queue is what determines when and how packets are dropped, transmitted, or delayed in accordance with configured bandwidth limitations (Xiaojing et al., 2012).

Implementing bandwidth control using the built-in Xen hypervisor traffic control tools certainly presents a straightforward approach to impose bandwidth limitations on virtual machines running within the hypervisor. The utilities within Xen however have two levels of granularity, unclassified traffic from the network interface and pre-configured traffic classifications used in traffic shaping (Xiaojing et al., 2012). While the concept of classifications of network bandwidth are similar to our bandwidth group structure, there are key differences. The classifications are static according to the configuration of the hypervisor, whereas bandwidth groups are dynamic and are created by the tenant. Traffic within a classification may be given best-effort treatment of bandwidth limitations which is similar to our work of traffic within a bandwidth group. The key difference however between the Xen hypervisor traffic control mechanism and out work of bandwidth groupings is the guaranteed maximum of bandwidth limitation afforded by Xen in contrast to the guaranteed minimum bandwidth allotment provided by this work of research.

To further demonstrate the differences, bandwidth test measurements are provided here with the same test parameters as described in section V regarding data

sent by a client virtual machine with outgoing client bandwidth limited by the receiving server. The software presented as part of this research is disabled to allow only the effects of the Xen bandwidth limitations to be in effect. Xen is configured to limit the bandwidth of the client virtual machine to 100,000 bytes per second, dropping packets that attempt to breach this limit.

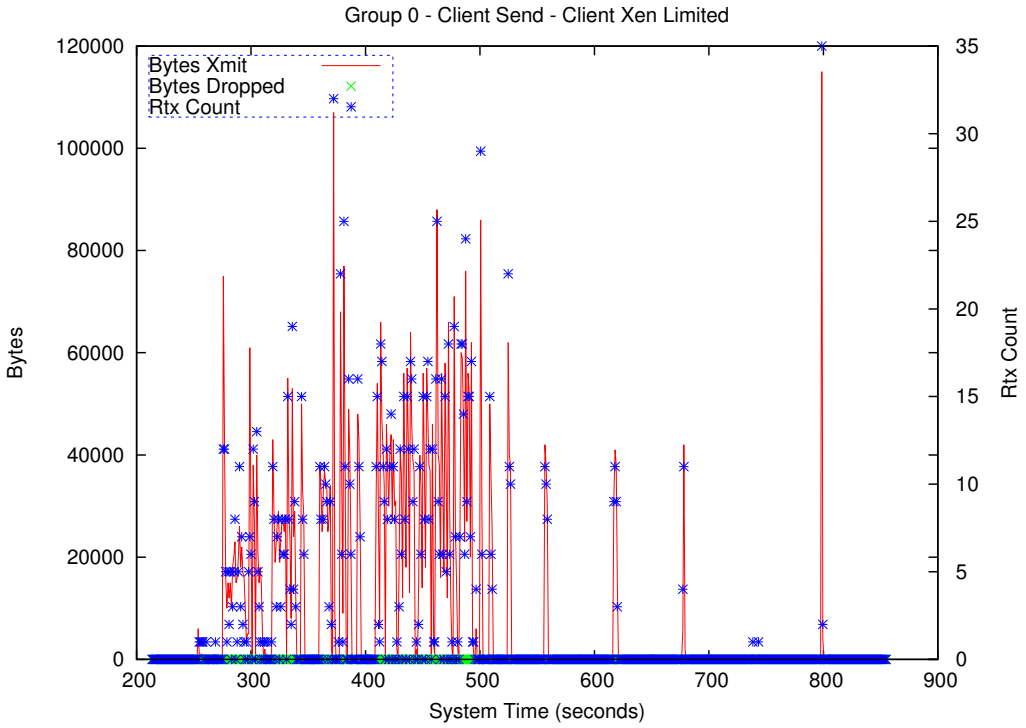


Figure V.19: Outgoing Bandwidth Connection 0

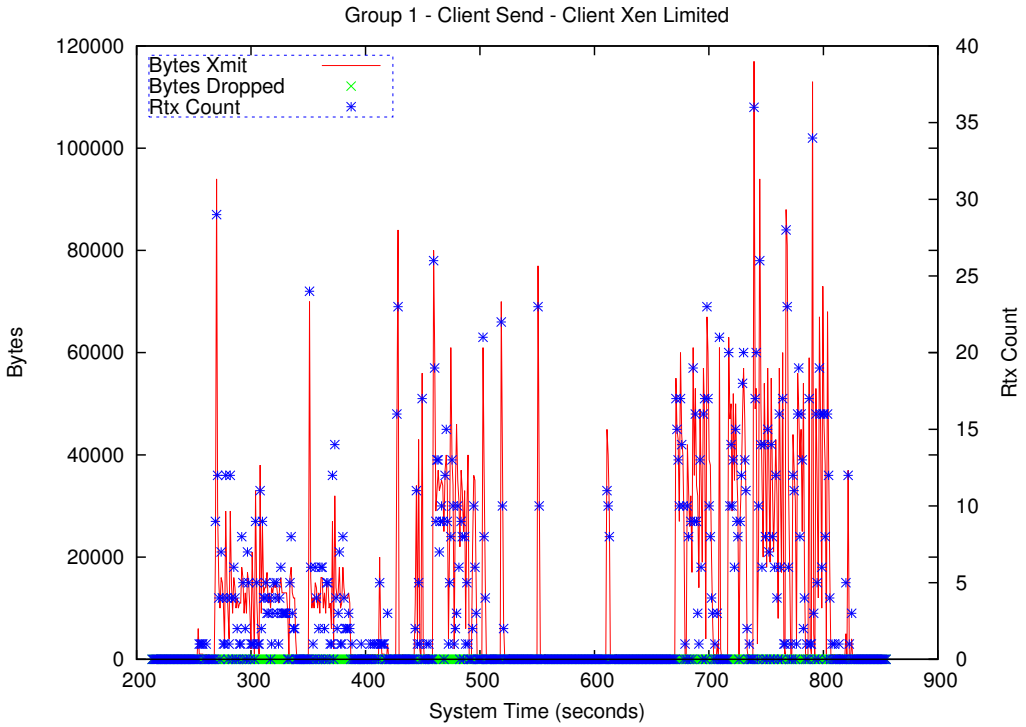


Figure V.20: Outgoing Bandwidth Connection 1

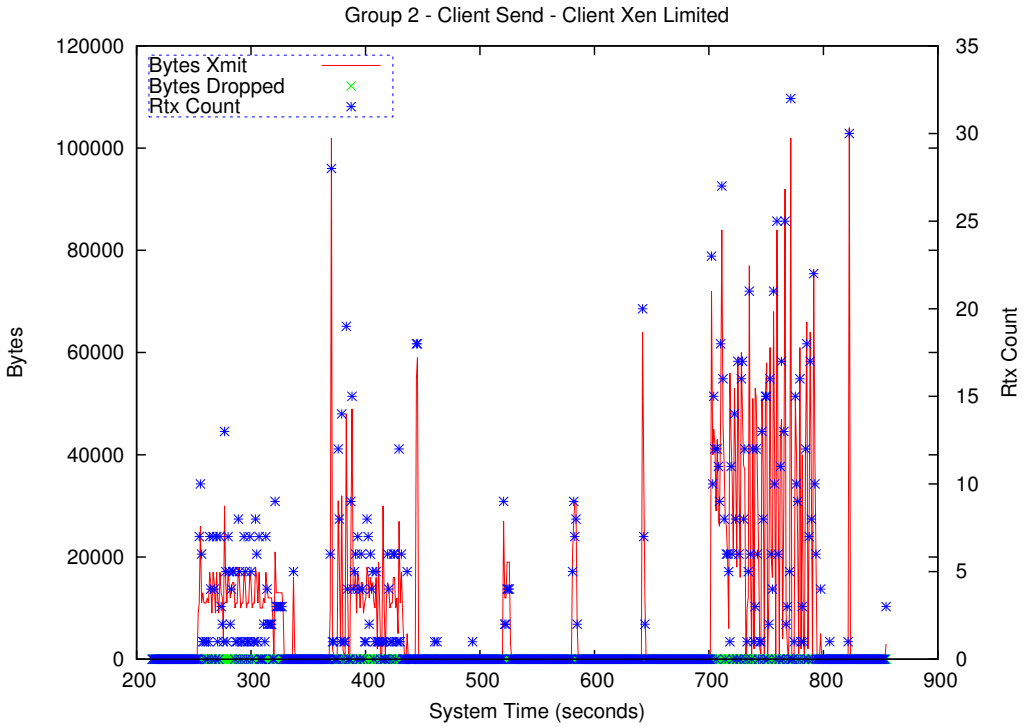


Figure V.21: Outgoing Bandwidth Connection 2

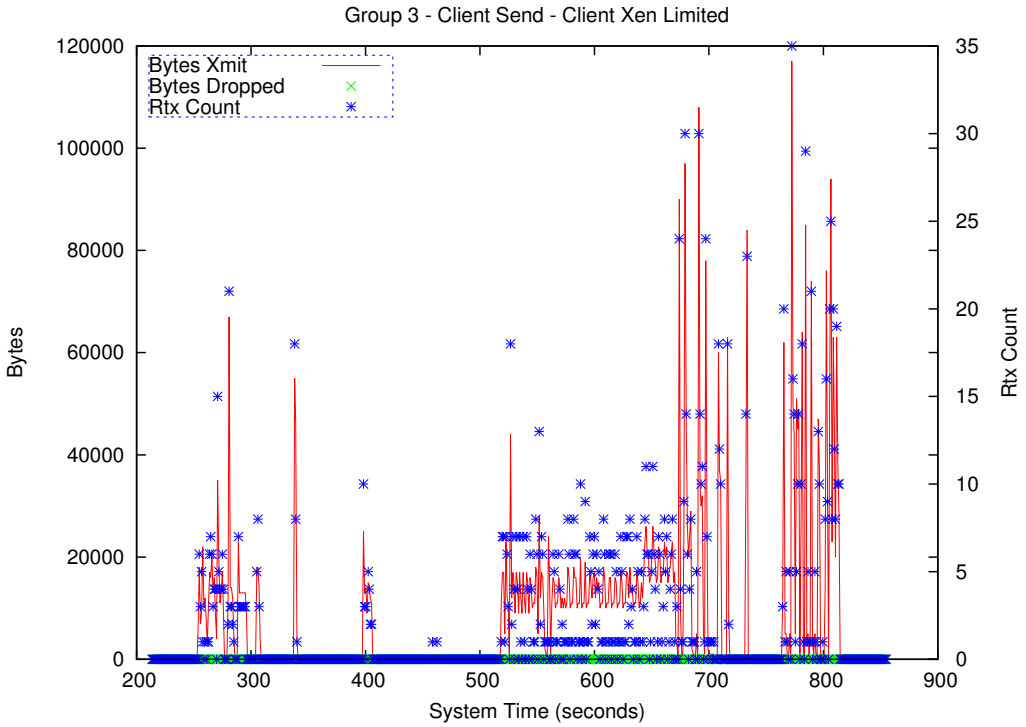


Figure V.22: Outgoing Bandwidth Connection 3

We consider the results of this test in figures V.19, V.20, V.21, and V.22 collectively, where each figure represents a discrete SCTP connection from client VM to server VM as conducted in section V. The pattern of traffic reveals a 'bursty' and stochastic behavior riddled with retransmissions and periods of time where little to no bandwidth is utilized by the four connections. This behavior is not wholly unexpected and demonstrates the key issue this research attempts to resolve.

VI. VIRTUAL SWITCH PERFORMANCE

When analyzing the performance of the bandwidth control algorithm for this thesis, questions arose regarding the nature of CPU utilization and throughput when virtual machines engage in network communications with each other. Previous analyses in prior works imply that the routing of packets by virtual switches is implemented in an inefficient manner, leading to an increase in CPU utilization as bandwidth load increases (Menon et al., 2005; Cherkasova and Gardner, 2005). Bardgett and Zou state that there are other approaches to the built-in Xen Bridge utilized in our work, such as Open vSwitch (Bardgett and Zou, 2012). In an effort to evaluate the performance impact of the virtual switching mechanism within our test bed, we catalog the performance in terms of CPU and bandwidth throughput for our test system for the built-in Xen Bridge and the Open vSwitch technologies. Additionally, other throughput performance enhancing techniques are discussed to provide a future direction for improving network I/O communications within the hypervisor.

Performance Test Setup

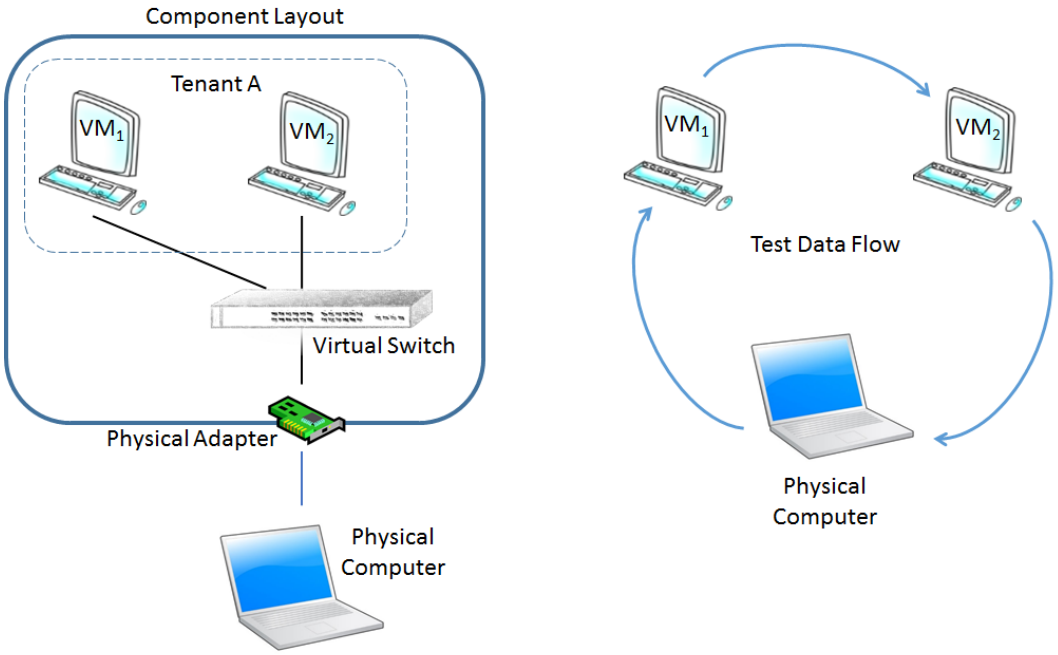


Figure VI.1: Virtual Machine Test Layouts

In our bandwidth performance test setup we have two virtual machines (VM1, VM2) hosted within the Xen hypervisor and a physical machine (PM) physically connected to the network port of the host machine. The virtual machines hosted within the hypervisor utilize the virtual switch for communications between each other and communications with the outside physical network. Our test setup utilizes iperf3 to perform network connectivity tests in a ring formation such that data flows from VM1 to VM2 to PM and back to VM1, as shown in figure VI.1. All increases in bandwidth for test purposes occur identically for all connections between machines. For all performance measurements, the impact imposed on the system by both the built-in Xen Bridge and Open vSwitch (in a mutually exclusive fashion) is cataloged.

CPU Performance

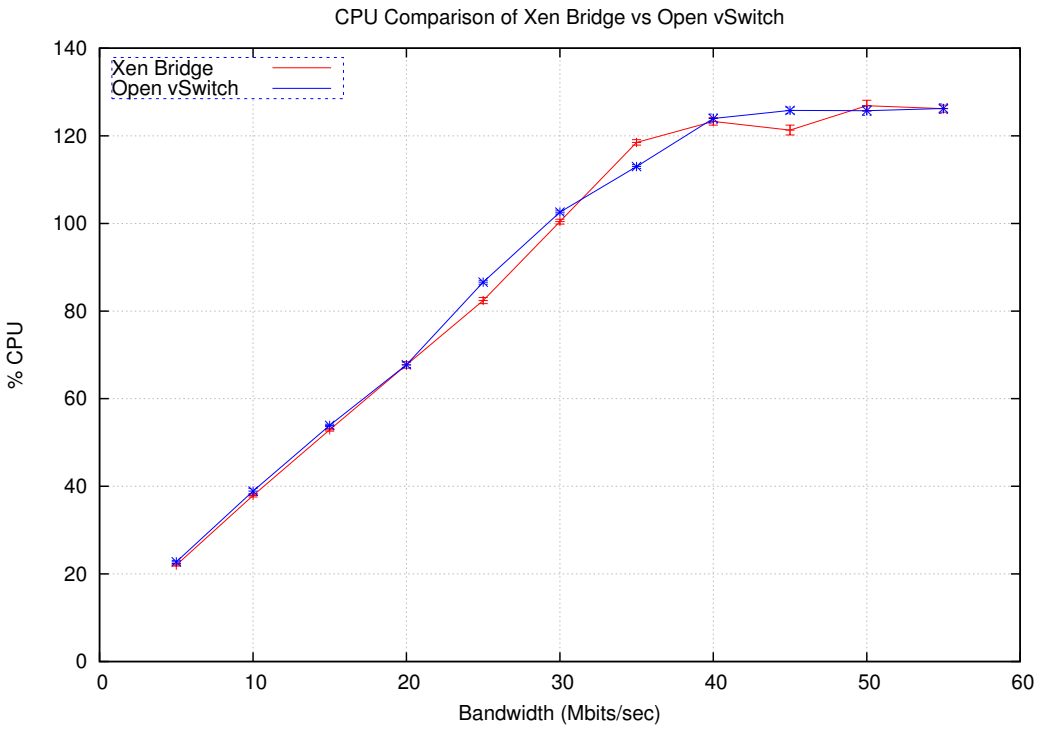


Figure VI.2: CPU Utilization vs Bandwidth

This section describes CPU utilization for dom0 of the Xen Hypervisor as bandwidth between the test machines increases linearly. The goal of this test is to profile CPU utilization of the overhead introduced by the incremental traffic between the virtual machines and the external physical host. We see in figure VI.2 that indeed CPU utilization increases as the bandwidth between machines increases, up to a limit, upon which CPU utilization levels off. Note that since the Xen Hypervisor host is running on a quad-core system, CPU utilization is represented in a scaling fashion from 0 - 400%, or 100% per core. We also note here that there is a limit around 40 Mbit per second where increased bandwidth utilization will no longer result in an increase in CPU utilization. This replicates the findings by Cherkasova and Gardner, with allowances of physical test system differences (Cherkasova and Gardner, 2005). This implies that the increased traffic beyond this point does not cause a demand on processing resources, suggesting lost data traffic.

VM1 to VM2 Bandwidth Throughput

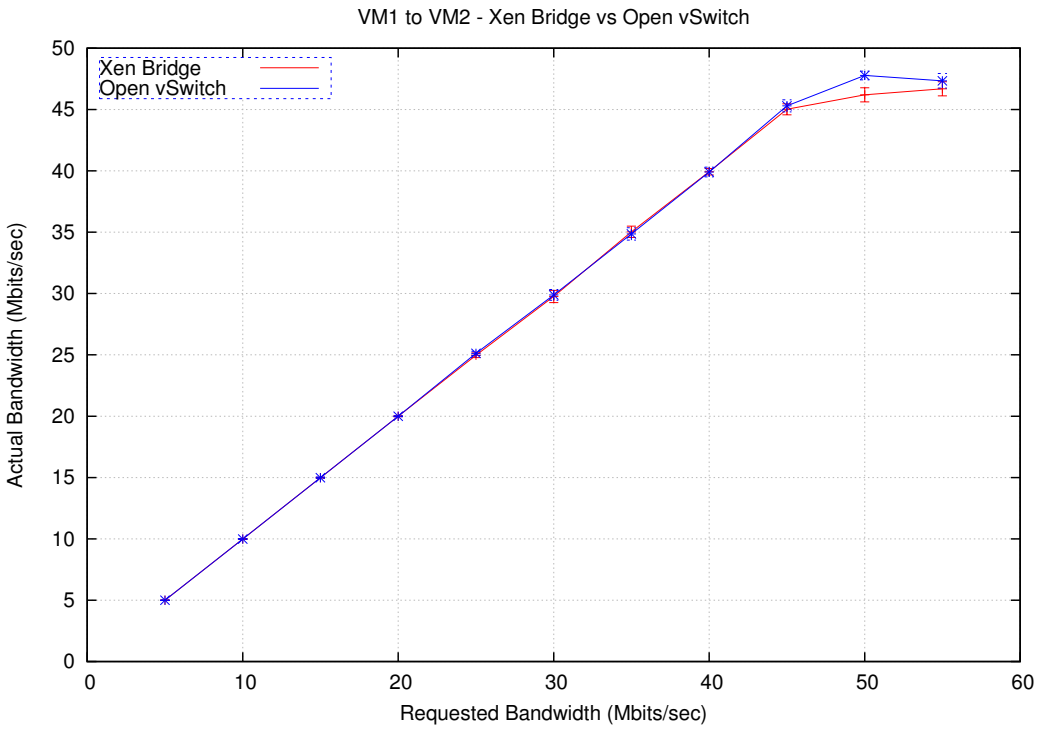


Figure VI.3: VM1 to VM2 - Confirmed Bandwidth vs Requested Bandwidth

This section describes the throughput between VM1 to VM2 as described in figure VI.3 as bandwidth increases across all machines. We see as the requested bandwidth sent by the iperf3 tool increases the confirmed bandwidth increases linearly to a operational limit of approximately 45 Mbit per second. At this point the throughput between VM1 and VM2 begins to level off. This test shows that Open vSwitch allows for slightly more bandwidth in the 50 Mbit per second case, but both fall short of the requested bandwidth. This confirms Bardgett and Zou's findings regarding the limited success of Open vSwitch (Bardgett and Zou, 2012).

VM2 to PM Bandwidth Throughput

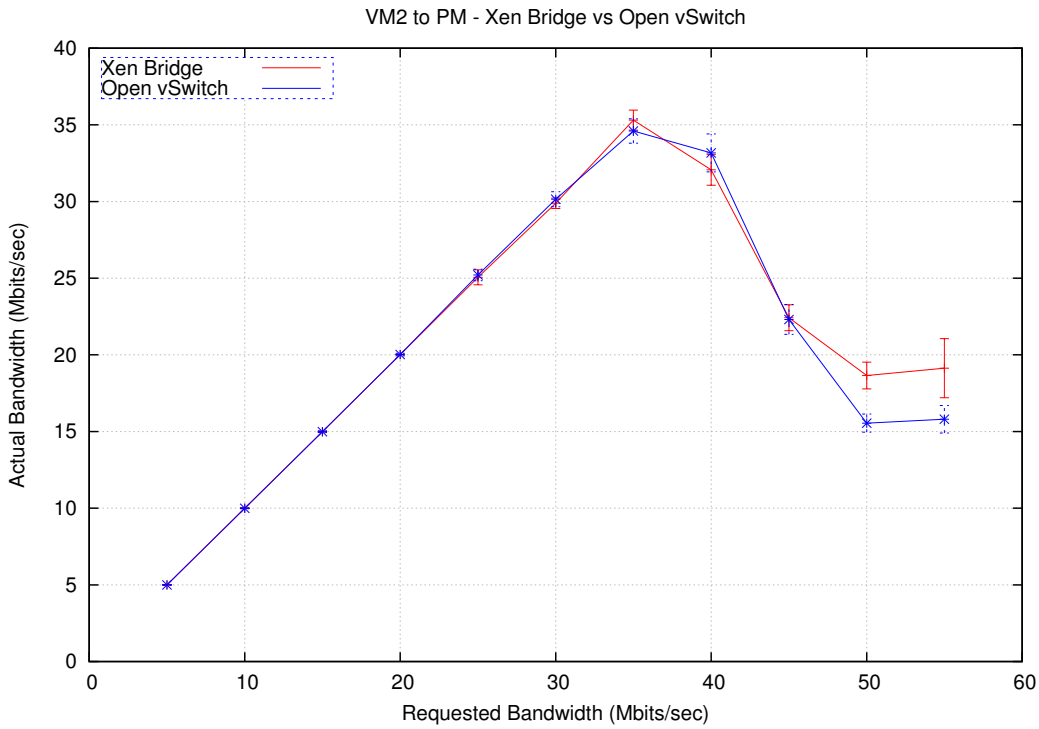


Figure VI.4: VM2 to PM - Confirmed Bandwidth vs Requested Bandwidth

This section describes the throughput between VM2 to the PM as described in figure VI.4 as bandwidth increases across all machines. We see as the requested bandwidth sent by the iperf3 tool increases the confirmed bandwidth increases linearly to a operational limit of approximately 35 Mbit per second. At this point the throughput between VM1 and VM2 begins to drop off significantly for both Xen Bridge and Open vSwitch. This test shows that Xen Bridge allows for slightly more bandwidth in the 50 Mbit per second case, but both fall short of the requested bandwidth. While certainly anomalous, these findings confirm the measurements documented by Menon, *et al* for throughput between virtual machines in Xen, allowing for differences between test systems (Menon et al., 2005).

PM to VM1 Bandwidth Throughput

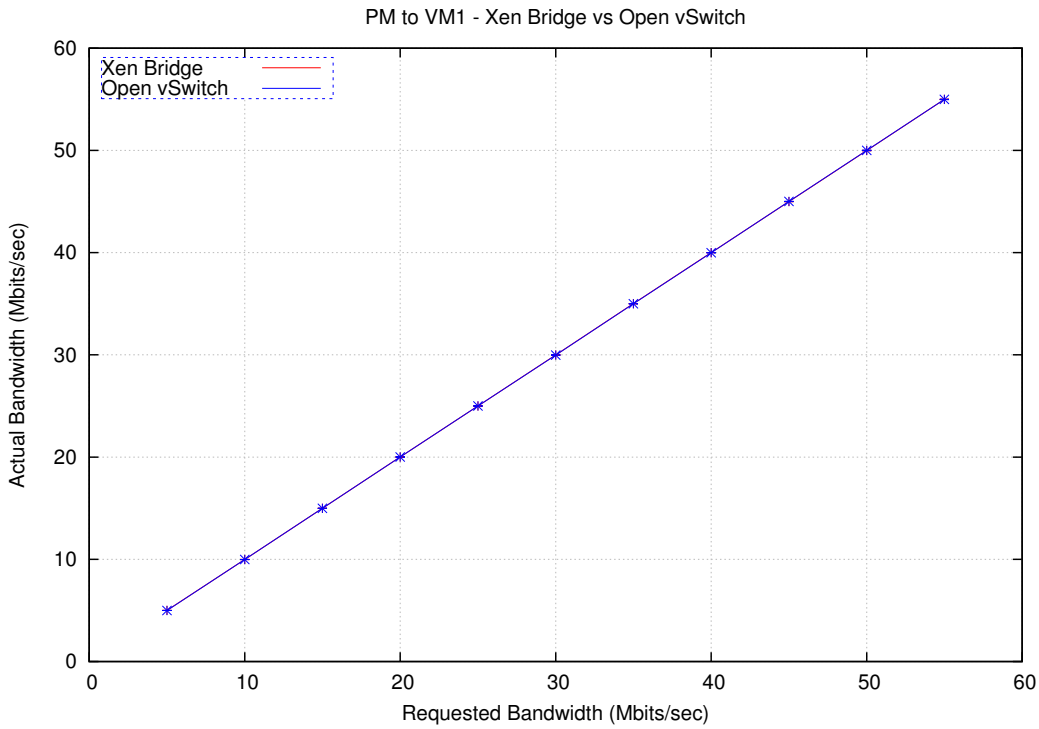


Figure VI.5: PM to VM1 - Confirmed Bandwidth vs Requested Bandwidth

This section describes the throughput between the PM to VM1 as described in figure VI.5 as bandwidth increases across all machines. We see as the requested bandwidth sent by the iperf3 tool increases the confirmed bandwidth increases linearly for the entire test case, up to 55 Mbit per second. It is quite interesting that incoming traffic from the physical machine to the virtual machine proceeds without any performance degradation. This finding implies that traffic incoming from the physical network proceeds with much higher throughput and efficiency as compared to traffic flowing between virtual machines. This also implies that software switching solutions (such as Open vSwitch and Xen Bridge) suffer from inefficiency issues, and other approaches such as SR-IOV may provide better alternatives (Bardgett and Zou, 2012). The following section discusses approaches towards pure software switching within the hypervisor.

Approaches to Hypervisor Virtual Switching

Xen Bridge

Xen Bridge, the software bridge that ships by default with the Xen hypervisor, is a mechanism where virtual machines may have their virtual NIC connected to the hosts physical NIC at the Ethernet address layer. This implies that virtual machine NICs may share the resources of the physical NIC for outside communications, and also for internal communications between virtual NICs. To accomplish this, the software bridge must keep track of traffic incoming and outgoing from the virtual machine interfaces, and forward traffic accordingly between the physical NIC and the virtual NICs, or between virtual NICs themselves. This technology forms the foundation of network virtualization within the hypervisor and is heavily built upon by other virtual switching technologies, which we discuss. Since the network bridge is implemented in software, a sizable overhead is encountered as traffic increases, and the number of virtual machines also increase (Luo, 2010).

XenLoop

The performance penalty associated with virtual switching is mainly borne by traffic moving from the source virtual machine, through the hypervisor (dom0 in Xen), and to the target virtual machine. This results in a large amount of performance degradation. Wang, *et al*, propose XenLoop, a technology which implements a VM-to-VM loopback channel, allowing network traffic between virtual machines to bypass existing hypervisor switching, increasing efficiency and performance. A primary advantage of XenLoop is that no code modifications are necessary to the guest virtual machines; only a kernel module is loaded in the guest virtual machines which manages 'channels' between virtual machines, avoiding the complexity of the virtual switch. The kernel module intercepts packets at the network layer using netfilter to determine source and destination, using this information to dynamically populate a table of machines used to discover other virtual ma-

chines. To accomplish the creation of XenLoop channels, the guest VM kernel module must know which other virtual machines are willing to setup channels. Xen has a facility called 'XenStore' which maintains tables of information specific to guest virtual machines running on the hypervisor. The guest VM being an unprivileged domain in the Xen hypervisor may only modify the contents of it's own entry in the XenStore; therefore it is unable to discover any other virtual machine entries. The guest VM creates a 'XenLoop' entity within its own XenStore. A 'domain discovery' module which runs in Dom0 (a privileged domain) is responsible for scanning the XenStore periodically, and upon discovery of a virtual machine advertising their ability for a XenLoop connection, will send a specially crafted network frame containing a list of all local virtual machines that are capable of XenLoop functionality. The guest VMs wishing to create a channel between themselves for communication utilize shared memory in a FIFO configuration in three regions. One region for incoming traffic, one for outgoing traffic, and a third for event notification. These regions are created using a facility for inter-domain shared memory. With this mechanism, XenLoop decreases packet latency by a factor of five, and increases bandwidth by a factor of six (Wang et al., 2008).

XenSockets

In marshaling communications between virtual machines within the same host, Xen utilizes a page flipping mechanism whereby network data within a system defined memory page is swapped by the hypervisor from one VM to the other, resulting in a zero-copy method of data transmission between virtual machines. Zhang, *et al*, surmise that a significant source of communications inefficiency between virtual machines sharing the same host lies in the TCP/IP stack over-utilizing the page flipping feature of Xen whereby low throughput situations and many page flipping calls by the TCP/IP stack results in significantly lower performance. XenSockets is an approach to solving the inter-VM performance problem by implementing a socket layer within the guest virtual machines and utilizing shared

memory for data transmission. Within the shared memory regions of XenSocket are circular buffers which are utilized in a producer-consumer pattern in FIFO orientation. This substitutes the zero-copy, page-flipping mechanism with a copy into the shared region by the data producer VM and a copy out of shared memory by a data consumer VM. The network API to both VM's is unchanged. Zhang, *et al*, observe that when the message sizes are large, XenSockets achieve large performance improvements when compared against the native page-flipping mechanism. They observe that the biggest impacts to performance are the per-message overhead for many smaller messages, and the data transfer overhead for many large messages. Their work results in a seventy-two times improvement for TCP streams with common message sizes (16 KB messages) (Zhang et al., 2007).

Direct I/O

Network I/O virtualization can be plagued with high overhead and incomplete emulations. As a result, devices shared by virtual machines cannot execute I/O device functionality anywhere near the performance of a native (physical) solution. This is invariably the case when virtual machines running within a cloud environment naturally share host system resources. Dong, *et al*, present Direct I/O, a technology which allows the hypervisor to present the same hardware semantics to the virtual machine as the physical host in terms of interrupt access, device register access and shared memory (DMA) access. When a device is directly assigned to a virtual machine, the VM may directly access the device as though it was running in a native environment, without much of the performance degradation caused by emulation in the hypervisor. With Direct I/O devices are singly assigned to a VM, and assignments may change between VMs. This naturally causes issues with device state being maintained when switched between virtual machines; Dong, *et al*, propose resetting the device through PCI-level reset commands to prevent state holdover. Direct I/O with devices that share interrupts presents a challenge because the native host hardware does not immediately know which driver will

service the interrupt between devices. Additionally, the hypervisor also does not immediately know, so their work focuses on utilizing Message Shared Interrupt, a hardware level technology which assigns dedicated physical interrupts to devices which would normally share interrupt lines, hence nullifying the shared IRQ concern. Cache coherency is also a concern whereby memory that is mapped between the virtual machine and the device shared memory region (DMA) must be of the same type in both the hypervisor and the virtual machine. Otherwise if the virtual machine believes the memory to be uncached, while the device is utilizing cached memory, then the MMU may not flush the memory appropriately from cache to RAM, causing data change to be potentially invisible by either entity. Their work makes use of the IA-32 architecture which can dictate the region memory type to allow for consistent cache coherency actions between the physical device and the virtual machine. Device drivers running within the VM may need to be modified to have knowledge that their previously native hardware interface has been replaced with a Direct I/O implementation, due to what Dong, *et al*, refer to as "driver virtualization holes". These can occur when device specific functionality is not virtualized and could be considered missing to the driver running in the virtual machine, such as directly accessing the PCI bus registers along with PCI addressing inconsistencies between the VM and the physical host. Their work results in an improvement in virtual device performance up to 98% of native performance, and timing latency applications are improved up to 4.8x (Dong et al., 2009).

vFlood

A contributing factor in TCP transport performance is the difficulty of a virtual machine's TCP stack to respond to ACK packets from receiving endpoints along the transmit path in a timely fashion. Delays in scheduling virtual machines imposed by the hypervisor can cause latency orders of magnitude higher than if the stack were implemented in a native environment. Gamage, *et al*, propose vFlood, a mechanism which moves the congestion control mechanism of the TCP stack out

of the virtual machine environment and into the hypervisor environment. This leaves the virtual machine to "flood" the network driver with data since the hypervisor can respond to acknowledgments and retransmissions in a more efficient manner. Their work also implements flow control of the virtual machines to prevent over-utilization of driver buffers, a finite resource. Additionally, connection controls are implemented to prevent a flood of connections over-utilizing available buffers, affecting all virtual machine connections to the network (internal or external endpoints). Since all congestion control activities are moved into the hypervisor driver (dom0 in Xen), vFlood reflects an improvement in data throughput by 5x (Gamage et al., 2011).

Large Receive Offload

In attempting to solve the problems of TCP packet overhead as a general consideration, Menon and Zwaenepoel observe that per-packet overhead dominates the primary source of performance degradation. In an effort to reduce per-packet overhead they propose two solutions. First is what they term "packet aggregation" where multiple packets are condensed into a single large packet such that per-packet processing is reduced by the amount of aggregation occurring. Secondly, what they term "TCP acknowledgment offload" is when an acknowledgment template packet is preallocated and reused to avoid the buffer allocation for every ACK packet. These optimizations are similar to vFlood but the performance claims are very interesting. They claim 45%-67% improvement in native Linux, and an 86% improvement of a Linux virtual machine running in the Xen hypervisor. This solution is considered to scale well with many increasing connections (Menon and Zwaenepoel, 2008).

Open vSwitch

Open vSwitch is a pure-software implemented virtual switch which marshals networking traffic between virtual nodes within a host and physical nodes outside

the host, connected to the switched network. Open vSwitch has a kernel module component and a user space component, both of which implement special rules which can provide optimized paths for data flows. Emmerich, *et al*, categorize the performance of Open vSwitch against IP forwarding and Xen Bridge, showing that Open vSwitch is the fastest and most efficient switching mechanism that is fully implemented in software. An interesting observation by Emmerich, *et al* is that dropping packets due to network congestion of the virtual switch is CPU intensive, such that it can cause increase context switching when servicing packet queues. Additionally, the virtual machine is continuously switched to provide an opportunity for the VMs to retrieve received packets, causing further overhead and worse packet drop frequency (Emmerich et al., 2014).

Virtual Switching with Intelligent NIC

In an effort to offload switching operation from the kernel or user space implementations, Luo, *et al* propose network switching functionality implemented in specially equipped network interface cards which contain the switching logic within programmable cores. The core idea is to allow for all packet-switching tasks to be offloaded to a hardware acceleration engine within the network interface, freeing the hypervisor host CPU utilization from those packet-switching activities. A driver in the hypervisor marshals data between VM mapped memory regions and the PCI memory for the network interface. Multiple instances of network switch implementations may be hosted within a single network interface card. The unit under test (Netronome NFE-i8000 Network Acceleration Card) packet round trip times are improved with this technology as compared to software defined switching mechanisms up to 20% (Luo et al., 2009).

Virtual Machine Device Queues

Similar to virtual switching with intelligent NICs, Intel has created Virtual Machine Device Queues (VMDq) which allows processing network traffic from virtual

machines within the hypervisor to be offloaded to a network interface implementing VMDq (Intel 82598 10 Gigabit Ethernet Controller). VMDq allows for a queue to be assigned to a specific virtual machine by the hypervisor. The hypervisor is responsible for moving the data between the VM and the assigned virtual device queue and vice-versa, offloading much of the work of switching and header inspection to the network interface. Intel claims that network throughput on an ESX hypervisor went from 4.0 Gbps before VMDq to 9.2 Gbps after VMDq was applied (Chinni and Hiremane, 2007).

SR-IOV

Single-Root I/O Virtualization is an effort to offload packet switching overhead within the hypervisor by offloading switching and network device functionality to the hardware layer. Similar to VMDq, Direct I/O and virtual switching with intelligent NICs, SR-IOV is a PCI-SIG specification which almost negates hypervisor involvement when virtual machines receive or transmit data. An SR-IOV compliant network interface device will create PCI "virtual functions", each of which may be assigned to specific virtual machine instances within the hypervisor. The Guest VM maintains a VF driver which maintains knowledge for communicating with the VF device, hosted by the SR-IOV hardware. The data mappings between the VF and the VM are accomplished using the host hardware IOMMU, completely avoiding the hypervisor (dom0) when transmitting and receiving network traffic data. This is possible because the VF is hosted by the SR-IOV compliant device connected to the PCI bus, therefore the virtual machine maintains a DMA mapping to PCI memory, and what the VM believes is a dedicated network interface. The VFs within the SR-IOV network card share the physical functions of the card and perform all switch-related activities within the card, sharing the physical network layer with all assigned VFs. The SR-IOV card itself is managed by a PF driver hosted within the hypervisor for configuration and assignment activities. Tests performed on an SR-IOV capable device (Intel 82576 Gigabit

Ethernet Controller) show 9.57 Gbps (out of a maximum of 10 Gbps) with 193% CPU utilization, which is only 48% above a native implementation, as compared to a standard CPU utilization of 499% at the same network throughput without SR-IOV. Inter-VM communication uses also flows through the network interface card between VFs, with the only limiting factor being the FIFO queues and the bandwidth limitations of PCIe bus transactions (Dong et al., 2012).

VII. ISSUES AND CONCERNS

In this section we discuss interesting and anomalous results from the experiments conducted as part of this research.

Covetous Bandwidth Effect

In the test case results we mention that a handful of retransmissions occur even though the bandwidth utilized is fully within the assured minimum bandwidth for the group. This is evidence of an anomaly in the proposed algorithm whereby other connections within bandwidth groups utilizing all available bandwidth could disrupt other bandwidth groups for a single quantum of bandwidth period (one second). This can occur under the following conditions.

- The virtual machine has met the maximum bandwidth limitation
- Connection groups utilize all available bandwidth
- One or more bandwidth groups increase bandwidth utilization, still within their configured assured minimum bandwidth

Consider a bandwidth group which, in prior quantas[1] is not fully utilizing its configured bandwidth, while within the current quanta another bandwidth group attempts to go beyond its configured bandwidth allocation. The bandwidth controller utilizes the unused bandwidth from the first group to allow the second group additional bandwidth. Then, also occurring within the same quanta, the first bandwidth group attempts to utilize the remainder of its configured bandwidth, however there is now none available since the second group consumed all remaining bandwidth. The first group is therefore denied use of the bandwidth, and packets are dropped, causing the retransmission to occur for connections within the first bandwidth group. Under these conditions, the bandwidth groups are competing

[1]For this research, a quanta period is defined as one second

for available bandwidth up the total allowed for the virtual machine (coveting bandwidth beyond what was guaranteed) for that specific bandwidth quantum, not giving a chance for other bandwidth groups to utilize bandwidth within its assured minimum limit. This effect lasts for only one quantum and is corrected for in the following quantum.

There are some solutions to this issue. The quanta time may be shortened to minimize this effect. Bandwidth over-utilization could also be allowed for a maximum of one quanta period to minimize retransmissions.

Transport Congestion Window and RTX

The congestion window as used in SCTP is the amount of unacknowledged transmitted data per connection may be outstanding before the stack delays further packet transmissions. SCTP continuously increases the congestion window in attempting to find the maximum throughput capable for the network link utilized using an AIMD[2] approach (Stewart and Xie, 2002). The bandwidth controller will drop packets that attempt to go beyond the bandwidth limits (for connections within the limited bandwidth group), causing retransmissions to occur and causing the SCTP protocol stack to significantly decrease the congestion window. However on future transmissions, the amount of data sent across the connection increases up to the bandwidth enforcement limit within a few seconds. This implies that the AIMD increase in the congestion windows by the SCTP stack after an RTX may be too aggressive, causing more RTXs to occur unnecessarily.

One proposed solution to this issue would be to modify the transport layer to have knowledge of the bandwidth limitation, however it would seem that this issue would occur on a network connection with a constant low bandwidth threshold. More research is necessary to determine if this is intentional behavior or a bug in the SCTP protocol stack.

[2]Additive Increase, Multiplicative Decrease

Explicit Congestion Notification

Testing showed that ECN[3] had no significant impact on the performance of transmitted SCTP data. While it was confirmed that ECN operation was taking place, there are limitations on how much ECN can affect the overall bandwidth. For example, the congestion window reduction that occurs in response to congestion conditions is confined to approximately one RTT[4] (Ramakrishnan et al., 2001). Ye, et al also state that ECN not a mechanism for congestion avoidance regarding both endpoints (Ye et al., 2005), which suggests questions about its usefulness for out intention of controlling bandwidth utilization. Even so, we believe that the general approach has merit for future consideration since utilizing IP level flags to inform endpoints about congestion may provide a direct mechanism to limit bandwidth at each endpoint.

SCTP Streams and Head of Line Blocking

SCTP was chosen as the transport protocol used in these experiments. Early in experiment design, this work focused on individual streams within a single connection being contained within different bandwidth groups. The original idea was to focus on how streams are not subject to head-of-line blocking; if a packet for a stream was dropped, only packets for that stream queue up waiting for the retransmission acknowledgment to occur. This is unlike TCP; if a packet was pending acknowledgment, future packets would queue up for transmission until the first acknowledgment came in for the packet at the "head of the line" (Scharf and Kiesel, 2006). With SCTP, other packets assigned to other streams continue to transmit, even as one stream is blocked (Stewart and Xie, 2002). If bandwidth limitations are about to be exceeded, we hypothesized that packets for a stream trying to utilize all available bandwidth could be dropped without affecting the data transmission of other streams. The thought was to capitalize on streams

[3]explicit congestion notification
[4]round trip time for a packet

within bandwidth groups since it was implied they should not affect each other due to the no head-of-line blocking feature.

The data collected showed that if one stream had packets dropped for bandwidth enforcement, all streams would slow or stop their data bandwidth utilization, even if the other streams within their respective bandwidth groups did not violate their assured minimum bandwidth limits. We found that even though dropping packets on one stream does not trigger head-of-line blocking on another stream, the congestion window for the entire connection would drop significantly, thus affecting all streams within that connection.

This finding presents a major concern for the streaming concept within SCTP. The no head-of-line blocking feature takes place on a "per-stream" basis. The congestion window feature takes place on a "per-connection" basis. This has the effect, on average, of making the no head-of-line blocking feature useless for our application of bandwidth control since all streams are strongly affected by congestion window shrinkage due to the dropped packet by any stream. Scharf and Kiesel mention that other modes of transmission unique to SCTP may alleviate some issues such as transmission in unordered or partial-ordered modes (Scharf and Kiesel, 2006). It would be interesting to pursue this as a separate research problem, and perhaps attempting to solve it with a congestion window for each stream. More research is necessary to determine if this is intentional behavior or a bug in the SCTP protocol stack.

In response, the research focused instead on connections within bandwidth groups as opposed to streams within bandwidth groups. This resulted in the expected theoretical and experimental behavior shown in the sections above. As such, this work should have identical results for any reliable transport protocol with retransmission capabilities, such as TCP.

VIII. CONCLUSION

We propose "Bandwidth as a Service" as a beginning towards providing tenants control over guaranteed minimum bandwidths to VM network connections while the cloud provider maintains overall bandwidth control. We contribute to the BaaS concept through management of bandwidth groups that allow tenant virtual machines to have programmatic control over how network connections are managed, while overall bandwidth limitations are enforced via incoming and outgoing data rates. The cloud bandwidth controller serves as the agent which dictates how much bandwidth tenant VMs may consume. The value-add service we propose allows tenant VMs to specify to the cloud bandwidth controller sections of bandwidth that are dedicated to particular network connections. The experimental results shown here demonstrate the algorithm to perform this value-add service by a cloud bandwidth controller. We also examine the effects that the software switching solution has on network throughput, and provide discussion for future areas of research with alternative mechanisms for network throughput enhancements. Other areas for further research include examining in detail the relationship between congestion window and stream RTX as mentioned above, and applying this algorithm to other connection oriented transport protocols such as TCP. Examining the "Covetous Bandwidth Effect" should also be a focus of further research, in order to further refine the bandwidth control algorithm. Another step would be to apply this bandwidth controller concept to an actual bandwidth controller service within a cloud simulation to categorize the performance of cloud network bandwidth. As mentioned in section VII, SCTP was chosen as the vehicle to implement the bandwidth control mechanism of this research. This decision afforded privileged access to the lower levels of the Guest VM networking stack, providing an accurate means of evaluating our BaaS model. Consequently, we seek to implement this research at the hypervisor level in order to decouple the concept from any specific transport technology or operating system within guest VMs.

APPENDIX SECTION

APPENDIX A - MEASURED DATA

These data represent the experimental results for which this work is based.

A.1 Client Sends Data to Server with Bandwidth Limiting

Table A.1: Server Receiving Data - Server Incoming Bandwidth Limited

	Bandwidth (bytes per second)				Bytes Dropped				Retransmitted Packets			
Time Stamp	Group 0	Group 1	Group 2	Group 3	Group 0	Group 1	Group 2	Group 3	Group 0	Group 1	Group 2	Group 3
66.640020	0	0	0	0	0	0	0	0	0	0	0	0
67.643397	0	0	0	0	0	0	0	0	0	0	0	0
68.646703	0	0	0	0	0	0	0	0	0	0	0	0
69.650056	0	0	0	0	0	0	0	0	0	0	0	0
70.653370	0	0	0	0	0	0	0	0	0	0	0	0
71.656742	0	0	0	0	0	0	0	0	0	0	0	0
72.660047	0	0	0	0	0	0	0	0	0	0	0	0
73.663404	0	0	0	0	0	0	0	0	0	0	0	0
74.666711	0	0	0	0	0	0	0	0	0	0	0	0
75.670090	0	0	0	0	0	0	0	0	0	0	0	0
76.673387	0	0	0	0	0	0	0	0	0	0	0	0
77.676745	0	0	0	0	0	0	0	0	0	0	0	0
78.680038	0	0	0	0	0	0	0	0	0	0	0	0
79.683419	0	0	0	0	0	0	0	0	0	0	0	0
80.686712	0	0	0	0	0	0	0	0	0	0	0	0
81.690072	0	0	0	0	0	0	0	0	0	0	0	0
82.693386	0	0	0	0	0	0	0	0	0	0	0	0
83.696731	0	0	0	0	0	0	0	0	0	0	0	0
84.700065	0	0	0	0	0	0	0	0	0	0	0	0
85.703390	0	0	0	0	0	0	0	0	0	0	0	0
86.706682	0	0	0	0	0	0	0	0	0	0	0	0
87.710079	0	0	0	0	0	0	0	0	0	0	0	0
88.713391	0	0	0	0	0	0	0	0	0	0	0	0
89.716715	0	0	0	0	0	0	0	0	0	0	0	0
90.720058	0	0	0	0	0	0	0	0	0	0	0	0
91.723417	0	0	0	0	0	0	0	0	0	0	0	0
92.726696	0	0	0	0	0	0	0	0	0	0	0	0

93.730062	0	0	0	0	0	0	0	0	0	0	0	0
94.733386	0	0	0	0	0	0	0	0	0	0	0	0
95.736715	0	0	0	0	0	0	0	0	0	0	0	0
96.740049	0	0	0	0	0	0	0	0	0	0	0	0
97.743379	0	0	0	0	0	0	0	0	0	0	0	0
98.746685	0	0	0	0	0	0	0	0	0	0	0	0
99.750050	0	0	0	0	0	0	0	0	0	0	0	0
100.753380	0	0	0	0	0	0	0	0	0	0	0	0
101.756718	0	0	0	0	0	0	0	0	0	0	0	0
102.760048	0	0	0	0	0	0	0	0	0	0	0	0
103.763379	0	0	0	0	0	0	0	0	0	0	0	0
104.766698	0	0	0	0	0	0	0	0	0	0	0	0
105.770053	0	0	0	0	0	0	0	0	0	0	0	0
106.773385	0	0	0	0	0	0	0	0	0	0	0	0
107.776725	0	0	0	0	0	0	0	0	0	0	0	0
108.780054	0	0	0	0	0	0	0	0	0	0	0	0
109.783389	0	0	0	0	0	0	0	0	0	0	0	0
110.786727	0	0	0	0	0	0	0	0	0	0	0	0
111.790054	0	0	0	0	0	0	0	0	0	0	0	0
112.793379	0	0	0	0	0	0	0	0	0	0	0	0
113.796733	0	0	0	0	0	0	0	0	0	0	0	0
114.800043	0	0	0	0	0	0	0	0	0	0	0	0
115.803375	0	0	0	0	0	0	0	0	0	0	0	0
116.806713	0	0	0	0	0	0	0	0	0	0	0	0
117.810047	0	0	0	0	0	0	0	0	0	0	0	0
165.970044	10000	10000	10000	10000	0	0	0	0	0	0	0	0
166.973379	10000	10000	10000	10000	0	0	0	0	0	0	0	0
167.976715	10000	10000	10000	10000	0	0	0	0	0	0	0	0
168.980018	10000	10000	10000	10000	0	0	0	0	0	0	0	0
169.983379	10000	10000	10000	10000	0	0	0	0	0	0	0	0
170.986704	10000	10000	10000	10000	0	0	0	0	0	0	0	0
171.990033	10000	10000	10000	10000	0	0	0	0	0	0	0	0
172.993343	10000	10000	10000	10000	0	0	0	0	0	0	0	0
173.996717	10000	10000	10000	10000	0	0	0	0	0	0	0	0
175.000045	10000	10000	10000	10000	0	0	0	0	0	0	0	0
176.003380	10000	10000	10000	10000	0	0	0	0	0	0	0	0
177.006702	10000	10000	10000	10000	0	0	0	0	0	0	0	0
178.010050	10000	10000	10000	10000	0	0	0	0	0	0	0	0
179.013387	10000	10000	10000	10000	0	0	0	0	0	0	0	0
180.016711	10000	10000	10000	10000	0	0	0	0	0	0	0	0
181.020044	10000	10000	10000	10000	0	0	0	0	0	0	0	0
182.023370	10000	10000	10000	10000	0	0	0	0	0	0	0	0
183.026699	10000	10000	10000	10000	0	0	0	0	0	0	0	0
184.030045	10000	10000	10000	10000	0	0	0	0	0	0	0	0
185.033355	10000	10000	10000	10000	0	0	0	0	0	0	0	0
186.036709	10000	10000	10000	10000	0	0	0	0	0	0	0	0
187.040007	10000	10000	10000	10000	0	0	0	0	0	0	0	0
188.043372	10000	10000	10000	10000	0	0	0	0	0	0	0	0

189.046670	10000	10000	10000	10000	0	0	0	0	0	0	0	0
190.050085	10000	10000	10000	10000	0	0	0	0	0	0	0	0
191.053361	10000	10000	10000	10000	0	0	0	0	0	0	0	0
192.056703	10000	10000	10000	10000	0	0	0	0	0	0	0	0
193.060002	10000	10000	10000	10000	0	0	0	0	0	0	0	0
194.063390	10000	10000	10000	10000	0	0	0	0	0	0	0	0
195.066669	10000	10000	10000	10000	0	0	0	0	0	0	0	0
196.070040	15000	10000	10000	10000	0	0	0	0	0	0	0	0
197.073325	15000	10000	10000	10000	0	0	0	0	0	0	0	0
198.076724	15000	10000	10000	10000	0	0	0	0	0	0	0	0
199.080022	15000	10000	10000	10000	0	0	0	0	0	0	0	0
200.083392	15000	10000	10000	10000	0	0	0	0	0	0	0	0
201.086664	15000	10000	10000	10000	0	0	0	0	0	0	0	0
202.090036	15000	10000	10000	10000	0	0	0	0	0	0	0	0
203.093337	15000	10000	10000	10000	0	0	0	0	0	0	0	0
204.096672	15000	10000	10000	10000	0	0	0	0	0	0	0	0
205.100002	15000	10000	10000	10000	0	0	0	0	0	0	0	0
206.103409	15000	10000	10000	10000	0	0	0	0	0	0	0	0
207.106663	15000	10000	10000	10000	0	0	0	0	0	0	0	0
208.110038	15000	10000	10000	10000	0	0	0	0	0	0	0	0
209.113373	15000	10000	10000	10000	0	0	0	0	0	0	0	0
210.116748	15000	10000	10000	10000	0	0	0	0	0	0	0	0
211.120023	15000	10000	10000	10000	0	0	0	0	0	0	0	0
212.123354	15000	10000	10000	10000	0	0	0	0	0	0	0	0
213.126666	15000	10000	10000	10000	0	0	0	0	0	0	0	0
214.130074	15000	10000	10000	10000	0	0	0	0	0	0	0	0
215.133363	15000	10000	10000	10000	0	0	0	0	0	0	0	0
216.136701	15000	10000	10000	10000	0	0	0	0	0	0	0	0
217.140037	15000	10000	10000	10000	0	0	0	0	0	0	0	0
218.143369	15000	10000	10000	10000	0	0	0	0	0	0	0	0
219.146684	15000	10000	10000	10000	0	0	0	0	0	0	0	0
220.150034	15000	10000	10000	10000	0	0	0	0	0	0	0	0
221.153341	15000	10000	10000	10000	0	0	0	0	0	0	0	0
222.156696	15000	10000	10000	10000	0	0	0	0	0	0	0	0
223.160016	15000	10000	10000	10000	0	0	0	0	0	0	0	0
224.163375	15000	10000	10000	10000	0	0	0	0	0	0	0	0
225.166703	15000	10000	10000	10000	0	0	0	0	0	0	0	0
226.170044	20000	10000	10000	10000	0	0	0	0	0	0	0	0
227.173330	20000	10000	10000	10000	0	0	0	0	0	0	0	0
228.176709	20000	10000	10000	10000	0	0	0	0	0	0	0	0
229.179994	20000	10000	10000	10000	0	0	0	0	0	0	0	0
230.183372	20000	10000	10000	10000	0	0	0	0	0	0	0	0
231.186659	20000	10000	10000	10000	0	0	0	0	0	0	0	0
232.190028	20000	10000	10000	10000	0	0	0	0	0	0	0	0
233.193342	20000	10000	10000	10000	0	0	0	0	0	0	0	0
234.196705	20000	10000	10000	10000	0	0	0	0	0	0	0	0
235.200016	20000	10000	10000	10000	0	0	0	0	0	0	0	0
236.203366	20000	10000	10000	10000	0	0	0	0	0	0	0	0

237.206707	20000	10000	10000	10000	0	0	0	0	0	0	0	0
238.210041	20000	10000	10000	10000	0	0	0	0	0	0	0	0
239.213345	20000	10000	10000	10000	0	0	0	0	0	0	0	0
240.216714	20000	10000	10000	10000	0	0	0	0	0	0	0	0
241.220029	20000	10000	10000	10000	0	0	0	0	0	0	0	0
242.223380	20000	10000	10000	10000	0	0	0	0	0	0	0	0
243.226706	20000	10000	10000	10000	0	0	0	0	0	0	0	0
244.230033	20000	10000	10000	10000	0	0	0	0	0	0	0	0
245.233370	20000	10000	10000	10000	0	0	0	0	0	0	0	0
246.236710	20000	10000	10000	10000	0	0	0	0	0	0	0	0
247.240045	20000	10000	10000	10000	0	0	0	0	0	0	0	0
248.243379	20000	10000	10000	10000	0	0	0	0	0	0	0	0
249.246704	20000	10000	10000	10000	0	0	0	0	0	0	0	0
250.250065	20000	10000	10000	10000	0	0	0	0	0	0	0	0
251.253371	20000	10000	10000	10000	0	0	0	0	0	0	0	0
252.256703	20000	10000	10000	10000	0	0	0	0	0	0	0	0
253.260043	20000	10000	10000	10000	0	0	0	0	0	0	0	0
254.263363	20000	10000	10000	10000	0	0	0	0	0	0	0	0
255.266712	20000	10000	10000	10000	0	0	0	0	0	0	0	0
256.270033	25000	10000	10000	10000	0	0	0	0	0	0	0	0
257.273348	25000	10000	10000	10000	0	0	0	0	0	0	0	0
258.276727	25000	10000	10000	10000	0	0	0	0	0	0	0	0
259.280042	25000	10000	10000	10000	0	0	0	0	0	0	0	0
260.283366	25000	10000	10000	10000	0	0	0	0	0	0	0	0
261.286702	25000	10000	10000	10000	0	0	0	0	0	0	0	0
262.290057	25000	10000	10000	10000	0	0	0	0	0	0	0	0
263.293373	25000	10000	10000	10000	0	0	0	0	0	0	0	0
264.296704	25000	10000	10000	10000	0	0	0	0	0	0	0	0
265.300029	25000	10000	10000	10000	0	0	0	0	0	0	0	0
266.303362	25000	10000	10000	10000	0	0	0	0	0	0	0	0
267.306696	25000	10000	10000	10000	0	0	0	0	0	0	0	0
268.310031	25000	10000	10000	10000	0	0	0	0	0	0	0	0
269.313364	25000	10000	10000	10000	0	0	0	0	0	0	0	0
270.316699	25000	10000	10000	10000	0	0	0	0	0	0	0	0
271.319999	25000	10000	10000	15000	0	0	0	0	0	0	0	0
272.323362	25000	11000	19000	14000	0	0	0	0	0	0	0	0
273.326826	28000	18000	10000	10000	0	0	0	0	0	0	0	0
274.332409	45000	10000	10000	10000	0	0	0	0	0	0	0	0
275.333704	21000	10000	10000	10000	0	0	0	0	0	0	0	0
276.336652	31000	10000	10000	10000	0	0	0	0	0	0	0	0
277.339986	25000	10000	10000	10000	0	0	0	0	0	0	0	0
278.343321	25000	10000	10000	10000	0	0	0	0	0	0	0	0
279.346652	25000	10000	10000	10000	0	0	0	0	0	0	0	0
280.349990	25000	10000	10000	10000	0	0	0	0	0	0	0	0
281.353326	25000	10000	10000	10000	0	0	0	0	0	0	0	0
282.356662	25000	10000	10000	10000	0	0	0	0	0	0	0	0
283.359994	25000	10000	10000	10000	0	0	0	0	0	0	0	0
284.363320	25000	10000	10000	10000	0	0	0	0	0	0	0	0

285.366661	29000	10000	10000	10000	0	0	0	0	0	0	0	0
286.369984	30000	10000	10000	10000	0	0	0	0	0	0	0	0
287.373318	31000	10000	10000	10000	0	0	0	0	0	0	0	0
288.376663	29000	10000	10000	10000	0	0	0	0	0	0	0	0
289.380005	31000	10000	10000	10000	0	0	0	0	0	0	0	0
290.383350	29000	10000	10000	10000	0	0	0	0	0	0	0	0
291.386652	30000	10000	10000	10000	0	0	0	0	0	0	0	0
292.390030	31000	10000	10000	10000	0	0	0	0	0	0	0	0
293.393335	30000	10000	10000	10000	0	0	0	0	0	0	0	0
294.396664	30000	10000	10000	10000	0	0	0	0	0	0	0	0
295.400013	29000	10000	10000	10000	0	0	0	0	0	0	0	0
296.403357	31000	10000	10000	10000	0	0	0	0	0	0	0	0
297.406650	29000	10000	10000	10000	0	0	0	0	0	0	0	0
298.409995	30000	10000	10000	10000	0	0	0	0	0	0	0	0
299.413318	30000	10000	10000	10000	0	0	0	0	0	0	0	0
300.416671	31000	10000	10000	10000	0	0	0	0	0	0	0	0
301.420009	29000	10000	10000	10000	0	0	0	0	0	0	0	0
302.423382	30000	10000	10000	10000	0	0	0	0	0	0	0	0
303.426687	30000	10000	10000	10000	0	0	0	0	0	0	0	0
304.430023	30000	10000	10000	10000	0	0	0	0	0	0	0	0
305.433340	31000	10000	10000	10000	0	0	0	0	0	0	0	0
306.436664	30000	10000	10000	10000	0	0	0	0	0	0	0	0
307.440014	30000	10000	10000	10000	0	0	0	0	0	0	0	0
308.443379	30000	10000	10000	10000	0	0	0	0	0	0	0	0
309.446682	30000	10000	10000	10000	0	0	0	0	0	0	0	0
310.450051	29000	10000	10000	10000	0	0	0	0	0	0	0	0
311.453366	31000	10000	10000	10000	0	0	0	0	0	0	0	0
312.456668	29000	10000	10000	11000	0	0	0	0	0	0	0	0
313.459994	30000	10000	10000	9000	0	0	0	0	0	0	0	0
314.463354	30000	10000	10000	10000	0	0	0	0	0	0	0	0
315.466668	30000	16000	10000	10000	0	0	0	0	0	0	0	0
316.470046	30000	15000	10000	10000	0	0	0	0	0	0	0	0
317.473340	30000	15000	10000	10000	0	0	0	0	0	0	0	0
318.476692	31000	15000	10000	10000	0	0	0	0	0	0	0	0
319.480005	29000	15000	10000	10000	0	0	0	0	0	0	0	0
320.483338	30000	15000	10000	10000	0	0	0	0	0	0	0	0
321.486655	30000	15000	10000	10000	0	0	0	0	0	0	0	0
322.489998	30000	15000	10000	10000	0	0	0	0	0	0	0	0
323.493324	31000	15000	10000	10000	0	0	0	0	0	0	0	0
324.496698	29000	15000	10000	10000	0	0	0	0	0	0	0	0
325.499991	30000	15000	10000	10000	0	0	0	0	0	0	0	0
326.503388	30000	15000	10000	10000	0	0	0	0	0	0	0	0
327.506690	31000	15000	10000	10000	0	0	0	0	0	0	0	0
328.510025	29000	15000	10000	10000	0	0	0	0	0	0	0	0
329.513345	30000	15000	10000	10000	0	0	0	0	0	0	0	0
330.516757	30000	15000	10000	10000	0	0	0	0	0	0	0	0
331.520029	30000	15000	10000	10000	0	0	0	0	0	0	0	0
332.523381	30000	15000	10000	10000	0	0	0	0	0	0	0	0

333.526844	30000	15000	10000	10000	0	0	0	0	0	0	0	0
334.530162	30000	15000	10000	11000	0	0	0	0	0	0	0	0
335.533464	30000	15000	11000	10000	0	0	0	0	0	0	0	0
336.536657	30000	15000	10000	10000	0	0	0	0	0	0	0	0
337.539977	30000	15000	10000	10000	0	0	0	0	0	0	0	0
338.543466	30000	15000	10000	10000	0	0	0	0	0	0	0	0
339.546661	31000	15000	10000	10000	0	0	0	0	0	0	0	0
340.549980	30000	15000	10000	10000	0	0	0	0	0	0	0	0
341.553330	30000	15000	10000	10000	0	0	0	0	0	0	0	0
342.556652	30000	15000	10000	10000	0	0	0	0	0	0	0	0
343.559997	30000	15000	10000	10000	0	0	0	0	0	0	0	0
344.563330	30000	15000	10000	10000	0	0	0	0	0	0	0	0
345.566660	30000	20000	10000	10000	0	0	0	0	0	0	0	0
346.570009	30000	20000	10000	10000	0	0	0	0	0	0	0	0
347.573358	30000	20000	10000	10000	0	0	0	0	0	0	0	0
348.576653	30000	20000	10000	10000	0	0	0	0	0	0	0	0
349.579977	30000	20000	10000	10000	0	0	0	0	0	0	0	0
350.583320	30000	20000	10000	10000	0	0	0	0	0	0	0	0
351.586652	30000	20000	10000	10000	0	0	0	0	0	0	0	0
352.590001	30000	20000	10000	10000	0	0	0	0	0	0	0	0
353.593328	30000	20000	10000	10000	0	0	0	0	0	0	0	0
354.596694	30000	20000	10000	10000	0	0	0	0	0	0	0	0
355.600029	30000	20000	10000	10000	0	0	0	0	0	0	0	0
356.603371	30000	20000	10000	10000	0	0	0	0	0	0	0	0
357.606669	30000	20000	10000	10000	0	0	0	0	0	0	0	0
358.610042	30000	20000	10000	10000	0	0	0	0	0	0	0	0
359.613362	30000	20000	10000	10000	0	0	0	0	0	0	0	0
360.616685	30000	20000	10000	10000	0	0	0	0	0	0	0	0
361.620023	30000	20000	10000	10000	0	0	0	0	0	0	0	0
362.623373	30000	20000	10000	10000	0	0	0	0	0	0	0	0
363.626688	30000	20000	10000	10000	0	0	0	0	0	0	0	0
364.630023	30000	20000	10000	10000	0	0	0	0	0	0	0	0
365.633368	30000	20000	10000	10000	0	0	0	0	0	0	0	0
366.636715	30000	20000	10000	10000	0	0	0	0	0	0	0	0
367.640029	30000	20000	10000	10000	0	0	0	0	0	0	0	0
368.643375	30000	20000	10000	10000	0	0	0	0	0	0	0	0
369.646696	30000	20000	10000	10000	0	0	0	0	0	0	0	0
370.650016	30000	20000	10000	10000	0	0	0	0	0	0	0	0
371.653308	30000	20000	10000	10000	0	0	0	0	0	0	0	0
372.656725	30000	20000	10000	10000	0	0	0	0	0	0	0	0
373.659983	30000	20000	10000	10000	0	0	0	0	0	0	0	0
374.663386	30000	20000	10000	10000	0	0	0	0	0	0	0	0
375.666673	30000	25000	10000	10000	0	0	0	0	0	0	0	0
376.670005	30000	25000	10000	10000	0	0	0	0	0	0	0	0
377.673314	30000	25000	10000	10000	0	0	0	0	0	0	0	0
378.676701	30000	25000	10000	10000	0	0	0	0	0	0	0	0
379.679989	30000	25000	10000	10000	0	0	0	0	0	0	0	0
380.683317	30000	25000	10000	10000	0	0	0	0	0	0	0	0

381.686655	30000	25000	10000	10000	0	0	0	0	0	0	0	0
382.690009	30000	25000	10000	10000	0	0	0	0	0	0	0	0
383.693357	30000	25000	10000	10000	0	0	0	0	0	0	0	0
384.696726	30000	25000	10000	10000	0	0	0	0	0	0	0	0
385.700005	30000	25000	10000	10000	0	0	0	0	0	0	0	0
386.703343	30000	25000	10000	10000	0	0	0	0	0	0	0	0
387.706688	30000	25000	10000	10000	0	0	0	0	0	0	0	0
388.710056	30000	25000	10000	10000	0	0	0	0	0	0	0	0
389.713333	30000	25000	10000	10000	0	0	0	0	0	0	0	0
390.716687	30000	25000	10000	10000	0	0	0	0	0	0	0	0
391.720011	30000	25000	10000	10000	0	0	0	0	0	0	0	0
392.723367	30000	25000	10000	10000	0	0	0	0	0	0	0	0
393.726683	30000	25000	10000	10000	0	0	0	0	0	0	0	0
394.730033	30000	25000	10000	10000	0	0	0	0	0	0	0	0
395.733443	30000	25000	10000	10000	0	0	0	0	0	0	0	0
396.736812	30000	25000	10000	10000	0	0	0	0	0	0	0	0
397.740114	30000	25000	10000	10000	0	0	0	0	0	0	0	0
398.743318	30000	25000	10000	10000	0	0	0	0	0	0	0	0
399.746662	30000	25000	10000	10000	0	0	0	0	0	0	0	0
400.749994	30000	25000	10000	10000	0	0	0	0	0	0	0	0
401.753505	30000	25000	10000	10000	0	0	0	0	0	0	0	0
402.756654	30000	25000	10000	10000	0	0	0	0	0	0	0	0
403.759997	30000	25000	10000	10000	0	0	0	0	0	0	0	0
404.763331	30000	25000	10000	10000	0	0	0	0	0	0	0	0
405.766660	30000	30000	10000	10000	0	0	0	0	0	0	0	0
406.769994	30000	30000	10000	10000	0	0	0	0	0	0	0	0
407.773321	30000	30000	10000	10000	0	0	0	0	0	0	0	0
408.776659	30000	30000	10000	10000	0	0	0	0	0	0	0	0
409.779995	30000	30000	10000	10000	0	0	0	0	0	0	0	0
410.783321	30000	30000	10000	10000	0	0	0	0	0	0	0	0
411.786658	30000	30000	10000	10000	0	0	0	0	0	0	0	0
412.789995	30000	30000	10000	10000	0	0	0	0	0	0	0	0
413.793326	30000	30000	10000	10000	0	0	0	0	0	0	0	0
414.796686	30000	30000	10000	10000	0	0	0	0	0	0	0	0
415.800018	30000	30000	10000	10000	0	0	0	0	0	0	0	0
416.803349	30000	30000	10000	10000	0	0	0	0	0	0	0	0
417.806651	30000	30000	10000	10000	0	0	0	0	0	0	0	0
418.810014	30000	30000	10000	10000	0	0	0	0	0	0	0	0
419.813327	30000	30000	10000	10000	0	0	0	0	0	0	0	0
420.816686	30000	30000	10000	10000	0	0	0	0	0	0	0	0
421.820014	30000	30000	10000	10000	0	0	0	0	0	0	0	0
422.823351	30000	30000	10000	10000	0	0	0	0	0	0	0	0
423.826661	30000	30000	10000	10000	0	0	0	0	0	0	0	0
424.830016	30000	30000	10000	10000	0	0	0	0	0	0	0	0
425.833359	30000	30000	10000	10000	0	0	0	0	0	0	0	0
426.836678	30000	30000	10000	10000	0	0	0	0	0	0	0	0
427.840023	30000	30000	10000	10000	0	0	0	0	0	0	0	0
428.843349	30000	30000	10000	10000	0	0	0	0	0	0	0	0

429.846667	30000	30000	10000	10000	0	0	0	0	0	0	0	0
430.850008	30000	30000	10000	10000	0	0	0	0	0	0	0	0
431.853303	30000	30000	10000	10000	0	0	0	0	0	0	0	0
432.856687	30000	30000	10000	10000	0	0	0	0	0	0	0	0
433.859999	30000	30000	10000	10000	0	0	0	0	0	0	0	0
434.863347	30000	30000	10000	10000	0	0	0	0	0	0	0	0
435.866676	30000	30000	15000	10000	0	0	0	0	0	0	0	0
436.870014	30000	30000	15000	10000	0	0	0	0	0	0	0	0
437.873313	30000	30000	15000	10000	0	0	0	0	0	0	0	0
438.876691	30000	30000	15000	10000	0	0	0	0	0	0	0	0
439.879981	30000	30000	15000	10000	0	0	0	0	0	0	0	0
440.883359	30000	30000	15000	10000	0	0	0	0	0	0	0	0
441.886672	30000	30000	15000	10000	0	0	0	0	0	0	0	0
442.890005	30000	30000	15000	10000	0	0	0	0	0	0	0	0
443.893313	30000	30000	15000	10000	0	0	0	0	0	0	0	0
444.896664	30000	30000	15000	10000	0	0	0	0	0	0	0	0
445.900003	30000	30000	15000	10000	0	0	0	0	0	0	0	0
446.903343	30000	30000	15000	10000	0	0	0	0	0	0	0	0
447.906659	30000	30000	15000	10000	0	0	0	0	0	0	0	0
448.909969	30000	30000	15000	10000	0	0	0	0	0	0	0	0
449.913309	30000	30000	15000	10000	0	0	0	0	0	0	0	0
450.916672	30000	30000	15000	10000	0	0	0	0	0	0	0	0
451.920001	30000	30000	15000	10000	0	0	0	0	0	0	0	0
452.923378	30000	30000	15000	10000	0	0	0	0	0	0	0	0
453.926645	30000	30000	15000	10000	0	0	0	0	0	0	0	0
454.930052	30000	30000	15000	10000	0	0	0	0	0	0	0	0
455.933334	30000	30000	15000	10000	0	0	0	0	0	0	0	0
456.936682	30000	30000	15000	10000	0	0	0	0	0	0	0	0
457.940005	30000	30000	15000	10000	0	0	0	0	0	0	0	0
458.943342	30000	30000	15000	10000	0	0	0	0	0	0	0	0
459.946638	30000	30000	15000	10000	0	0	0	0	0	0	0	0
460.950010	30000	30000	15000	10000	0	0	0	0	0	0	0	0
461.953317	30000	30000	15000	10000	0	0	0	0	0	0	0	0
462.956675	30000	30000	15000	10000	0	0	0	0	0	0	0	0
463.959985	30000	30000	15000	10000	0	0	0	0	0	0	0	0
464.963350	30000	30000	15000	10000	0	0	0	0	0	0	0	0
465.966683	30000	30000	20000	10000	0	0	0	0	0	0	0	0
466.970016	30000	30000	20000	10000	0	0	0	0	0	0	0	0
467.973331	30000	30000	20000	10000	0	0	0	0	0	0	0	0
468.976681	30000	30000	20000	10000	0	0	0	0	0	0	0	0
469.980014	30000	30000	20000	10000	0	0	0	0	0	0	0	0
470.983352	30000	30000	20000	10000	0	0	0	0	0	0	0	0
471.986664	30000	30000	20000	10000	0	0	0	0	0	0	0	0
472.990013	30000	30000	20000	10000	0	0	0	0	0	0	0	0
473.993333	30000	30000	20000	10000	0	0	0	0	0	0	0	0
474.996684	30000	30000	20000	10000	0	0	0	0	0	0	0	0
475.999984	30000	30000	20000	10000	0	0	0	0	0	0	0	0
477.003311	30000	30000	20000	10000	0	0	0	0	0	0	0	0

478.006669	30000	30000	20000	10000	0	0	0	0	0	0	0	0
479.010015	30000	30000	20000	10000	0	0	0	0	0	0	0	0
480.013357	30000	30000	20000	10000	0	0	0	0	0	0	0	0
481.016668	30000	30000	20000	10000	0	0	0	0	0	0	0	0
482.019969	30000	30000	20000	10000	0	0	0	0	0	0	0	0
483.023350	30000	30000	20000	10000	0	0	0	0	0	0	0	0
484.026677	30000	30000	20000	10000	0	0	0	0	0	0	0	0
485.030013	30000	30000	20000	10000	0	0	0	0	0	0	0	0
486.033347	30000	30000	20000	10000	0	0	0	0	0	0	0	0
487.036694	30000	30000	20000	10000	0	0	0	0	0	0	0	0
488.039967	30000	30000	20000	10000	0	0	0	0	0	0	0	0
489.043349	30000	30000	20000	10000	0	0	0	0	0	0	0	0
490.046641	30000	30000	20000	10000	0	0	0	0	0	0	0	0
491.050036	30000	30000	20000	10000	0	0	0	0	0	0	0	0
492.053341	30000	30000	20000	10000	0	0	0	0	0	0	0	0
493.056681	30000	30000	20000	10000	0	0	0	0	0	0	0	0
494.059983	30000	30000	20000	10000	0	0	0	0	0	0	0	0
495.063355	30000	30000	20000	10000	0	0	0	0	0	0	0	0
496.066659	30000	30000	25000	10000	0	0	0	0	0	0	0	0
497.070011	30000	30000	25000	10000	0	0	0	0	0	0	0	0
498.073347	30000	30000	25000	10000	0	0	0	0	0	0	0	0
499.076680	30000	30000	25000	10000	0	0	0	0	0	0	0	0
500.080002	30000	30000	25000	10000	0	0	0	0	0	0	0	0
501.083359	30000	30000	25000	10000	0	0	0	0	0	0	0	0
502.086671	30000	30000	25000	10000	0	0	0	0	0	0	0	0
503.090043	30000	30000	25000	10000	0	0	0	0	0	0	0	0
504.093309	30000	30000	25000	10000	0	0	0	0	0	0	0	0
505.096686	30000	30000	25000	10000	0	0	0	0	0	0	0	0
506.100005	30000	30000	25000	10000	0	0	0	0	0	0	0	0
507.103356	30000	30000	25000	10000	0	0	0	0	0	0	0	0
508.106644	30000	30000	25000	10000	0	0	0	0	0	0	0	0
509.110028	30000	30000	25000	10000	0	0	0	0	0	0	0	0
510.113325	30000	30000	25000	10000	0	0	0	0	0	0	0	0
511.116663	30000	30000	25000	10000	0	0	0	0	0	0	0	0
512.119984	30000	30000	25000	10000	0	0	0	0	0	0	0	0
513.123349	30000	30000	25000	10000	0	0	0	0	0	0	0	0
514.126640	30000	30000	25000	10000	0	0	0	0	0	0	0	0
515.130007	30000	30000	25000	10000	0	0	0	0	0	0	0	0
516.133317	30000	30000	25000	10000	0	0	0	0	0	0	0	0
517.136678	30000	30000	25000	10000	0	0	0	0	0	0	0	0
518.139973	30000	30000	25000	10000	0	0	0	0	0	0	0	0
519.143336	30000	30000	25000	10000	0	0	0	0	0	0	0	0
520.146671	30000	30000	25000	10000	0	0	0	0	0	0	0	0
521.150004	30000	30000	25000	10000	0	0	0	0	0	0	0	0
522.153341	30000	30000	25000	10000	0	0	0	0	0	0	0	0
523.156669	30000	30000	25000	10000	0	0	0	0	0	0	0	0
524.160013	30000	30000	25000	10000	0	0	0	0	0	0	0	0
525.163339	30000	30000	25000	10000	0	0	0	0	0	0	0	0

526.166642	10000	20000	30000	10000	19000	9000	0	0	0	0	0	0
527.170003	10000	21000	30000	10000	12000	13000	0	0	19	9	0	0
528.173304	12000	21000	30000	10000	7000	9000	0	0	12	13	0	0
529.176671	0	22000	30000	10000	0	9000	0	0	0	9	0	0
530.179979	13000	22000	30000	10000	8000	9000	0	0	7	9	0	0
531.183299	15000	22000	30000	10000	8000	9000	0	0	8	9	0	0
532.186651	16000	1000	30000	10000	8000	0	0	0	8	1	0	0
533.189963	16000	23000	30000	10000	8000	9000	0	0	8	8	0	0
534.193315	1000	25000	30000	10000	0	9000	0	0	1	9	0	0
535.196681	18000	25000	30000	10000	9000	9000	0	0	7	9	0	0
536.199972	19000	25000	30000	10000	9000	9000	0	0	9	9	0	0
537.203302	21000	25000	30000	10000	9000	9000	0	0	9	9	0	0
538.206648	22000	1000	30000	10000	9000	0	0	0	9	1	0	0
539.209980	22000	27000	30000	10000	9000	11000	0	0	9	8	0	0
540.213346	1000	29000	30000	10000	0	11000	0	0	1	11	0	0
541.216679	23000	29000	30000	10000	9000	11000	0	0	8	11	0	0
542.219998	24000	29000	30000	10000	9000	11000	0	0	9	11	0	0
543.223323	26000	29000	30000	10000	11000	11000	0	0	9	11	0	0
544.226655	27000	1000	30000	10000	11000	0	0	0	11	1	0	0
545.230003	27000	30000	30000	10000	11000	11000	0	0	11	10	0	0
546.233330	1000	32000	30000	10000	0	11000	0	0	1	11	0	0
547.236683	27000	32000	30000	10000	11000	11000	0	0	10	11	0	0
548.239967	11000	20000	30000	10000	8000	9000	0	0	11	11	0	0
549.243290	0	22000	30000	10000	0	9000	0	0	0	9	0	0
550.246644	12000	1000	30000	10000	8000	0	0	0	8	1	0	0
551.250023	12000	23000	30000	10000	8000	9000	0	0	8	8	0	0
552.253301	12000	24000	30000	10000	8000	9000	0	0	8	9	0	0
553.256656	1000	26000	30000	10000	0	11000	0	0	1	9	0	0
554.260008	14000	26000	30000	10000	8000	11000	0	0	7	11	0	0
555.263360	15000	26000	30000	10000	8000	11000	0	0	8	11	0	0
556.266661	17000	1000	30000	15000	9000	0	0	0	8	1	0	0
557.270011	17000	28000	30000	15000	9000	11000	0	0	9	10	0	0
558.273338	17000	29000	30000	15000	9000	11000	0	0	9	11	0	0
559.276674	1000	31000	30000	15000	0	11000	0	0	1	11	0	0
560.279971	18000	31000	30000	15000	9000	11000	0	0	8	11	0	0
561.283364	19000	31000	30000	15000	9000	11000	0	0	9	11	0	0
562.286630	21000	31000	30000	15000	9000	11000	0	0	9	11	0	0
563.290015	22000	1000	30000	15000	9000	0	0	0	9	1	0	0
564.293304	22000	31000	30000	15000	9000	11000	0	0	9	10	0	0
565.296705	1000	31000	30000	15000	0	11000	0	0	1	11	0	0
566.299970	22000	31000	30000	15000	9000	11000	0	0	8	11	0	0
567.303335	22000	31000	30000	15000	9000	11000	0	0	9	11	0	0
568.306666	22000	31000	30000	15000	9000	11000	0	0	9	11	0	0
569.310001	22000	1000	30000	15000	9000	0	0	0	9	1	0	0
570.313322	22000	32000	30000	15000	9000	11000	0	0	9	10	0	0
571.316670	1000	21000	30000	15000	0	9000	0	0	1	11	0	0
572.319975	11000	21000	30000	15000	8000	9000	0	0	8	9	0	0
573.323337	13000	21000	30000	15000	8000	9000	0	0	8	9	0	0

574.326641	14000	21000	30000	15000	8000	9000	0	0	8	9	0	0
575.329999	16000	1000	30000	15000	8000	0	0	0	8	1	0	0
576.333307	16000	22000	30000	15000	8000	9000	0	0	8	8	0	0
577.336690	1000	23000	30000	15000	0	9000	0	0	1	9	0	0
578.339986	18000	23000	30000	15000	9000	9000	0	0	7	9	0	0
579.343331	20000	23000	30000	15000	9000	9000	0	0	9	9	0	0
580.346634	21000	23000	30000	15000	9000	9000	0	0	9	9	0	0
581.349997	23000	1000	30000	15000	9000	0	0	0	9	1	0	0
582.353302	23000	25000	30000	15000	9000	11000	0	0	9	8	0	0
583.356645	1000	26000	30000	15000	0	11000	0	0	1	11	0	0
584.360003	24000	26000	30000	15000	9000	11000	0	0	8	11	0	0
585.365722	26000	26000	30000	17000	11000	11000	0	8000	9	11	0	0
586.368537	11000	20000	30000	37000	8000	9000	5000	0	11	11	0	8
587.371091	0	1000	32000	20000	0	0	12000	0	0	1	5	0
588.373289	10000	22000	1000	20000	8000	9000	0	0	8	8	1	0
589.376624	10000	20000	31000	20000	8000	9000	11000	0	8	9	11	0
590.379958	1000	20000	32000	20000	0	9000	11000	0	1	9	11	0
591.383294	11000	20000	32000	21000	8000	9000	11000	0	7	9	11	0
592.386625	13000	20000	32000	20000	8000	9000	11000	0	8	9	11	0
593.389960	14000	1000	32000	19000	8000	0	11000	0	8	1	11	0
594.393319	14000	21000	1000	20000	8000	9000	0	0	8	8	1	0
595.396631	14000	21000	34000	21000	8000	9000	11000	0	8	9	10	0
596.399961	14000	21000	36000	19000	8000	9000	11000	0	8	9	11	0
597.403326	1000	21000	37000	20000	0	9000	11000	0	1	9	11	0
598.406628	16000	21000	37000	20000	9000	9000	11000	0	7	9	11	0
599.409960	18000	1000	37000	20000	9000	0	11000	0	9	1	11	0
600.413322	18000	23000	1000	20000	9000	9000	0	0	9	8	1	0
601.416654	18000	23000	38000	20000	9000	9000	12000	1000	9	9	10	0
602.419962	10000	20000	31000	21000	8000	9000	11000	0	9	9	12	1
603.423310	1000	20000	33000	20000	0	9000	11000	0	1	9	11	0
604.426627	11000	20000	33000	19000	8000	9000	11000	0	7	9	11	0
605.429957	12000	20000	33000	20000	8000	9000	11000	0	8	9	11	0
606.433311	14000	1000	1000	20000	8000	0	0	0	8	1	1	0
607.436665	10000	20000	35000	20000	8000	9000	11000	0	8	8	10	0
608.439960	10000	20000	31000	20000	8000	9000	11000	0	8	9	11	0
609.443335	1000	20000	33000	20000	0	9000	11000	0	1	9	11	0
610.446630	11000	20000	33000	20000	8000	9000	11000	0	7	9	11	0
611.449958	12000	20000	33000	20000	8000	9000	11000	0	8	9	11	0
612.453310	14000	1000	1000	20000	8000	0	0	0	8	1	1	0
613.456636	10000	20000	35000	20000	8000	9000	11000	0	8	8	10	0
614.459981	10000	20000	31000	20000	8000	9000	11000	0	8	9	11	0
615.463342	1000	20000	33000	26000	0	9000	11000	0	1	9	11	0
616.466628	11000	20000	33000	25000	8000	9000	11000	0	7	9	11	0
617.469955	12000	20000	33000	25000	8000	9000	11000	0	8	9	11	0
618.473311	14000	1000	1000	25000	8000	0	0	0	8	1	1	0
619.476659	10000	20000	35000	25000	8000	9000	11000	0	8	8	10	0
620.480003	10000	20000	31000	25000	8000	9000	11000	0	8	9	11	0
621.483336	1000	20000	33000	25000	0	9000	11000	0	1	9	11	0

622.486666	11000	20000	33000	25000	8000	9000	11000	0	7	9	11	0
623.489982	12000	20000	33000	25000	8000	9000	11000	0	8	9	11	0
624.493288	14000	1000	33000	25000	8000	0	11000	0	8	1	11	0
625.496654	14000	21000	1000	25000	8000	9000	0	0	8	8	1	0
626.499996	14000	21000	34000	25000	8000	9000	11000	0	8	9	10	0
627.503321	1000	21000	35000	25000	0	9000	11000	0	1	9	11	0
628.506633	16000	21000	35000	25000	9000	9000	11000	0	7	9	11	0
629.509977	18000	21000	35000	25000	9000	9000	11000	0	9	9	11	0
630.513290	11000	1000	30000	25000	8000	0	11000	0	9	1	11	0
631.516644	11000	21000	1000	25000	8000	9000	0	0	8	8	1	0
632.519977	11000	21000	31000	25000	8000	9000	11000	0	8	9	10	0
633.523328	1000	21000	33000	25000	0	9000	11000	0	1	9	11	0
634.526618	12000	21000	33000	25000	8000	9000	11000	0	7	9	11	0
635.529974	13000	21000	33000	25000	8000	9000	11000	0	8	9	11	0
636.533293	15000	1000	33000	25000	8000	0	11000	0	8	1	11	0
637.536651	15000	23000	1000	25000	8000	9000	0	0	8	8	1	0
638.539990	15000	23000	35000	25000	8000	9000	11000	0	8	9	10	0
639.543326	1000	23000	36000	25000	0	9000	11000	0	1	9	11	0
640.546629	16000	23000	36000	24000	9000	9000	11000	1000	7	9	11	0
641.549994	11000	20000	30000	25000	8000	9000	11000	0	9	9	11	1
642.553298	13000	1000	30000	26000	8000	0	11000	0	8	1	11	0
643.556659	13000	21000	1000	25000	8000	9000	0	0	8	8	1	0
644.559970	13000	21000	31000	25000	8000	9000	11000	0	8	9	10	0
645.563336	1000	21000	32000	29000	0	9000	11000	0	1	9	11	0
646.566627	14000	21000	32000	30000	8000	9000	11000	0	7	9	11	0
647.569996	14000	21000	32000	31000	8000	9000	11000	0	8	9	11	0
648.573512	14000	1000	32000	30000	8000	0	11000	0	8	1	11	0
649.576914	14000	22000	1000	30000	8000	9000	0	0	8	8	1	0
650.579998	14000	22000	32000	30000	8000	9000	11000	0	8	9	10	0
651.583429	1000	22000	32000	30000	0	9000	11000	0	1	9	11	0
652.587519	15000	22000	32000	30000	10000	9000	11000	0	9	9	11	0
653.589956	0	20000	31000	30000	0	9000	11000	0	0	9	11	0
654.593283	11000	1000	31000	30000	8000	0	11000	0	8	1	11	0
655.596661	11000	21000	1000	30000	8000	9000	0	0	8	8	1	0
656.599973	1000	21000	32000	30000	0	9000	11000	0	1	9	10	0
657.603294	13000	21000	32000	30000	8000	9000	11000	0	7	9	11	0
658.606620	15000	21000	32000	30000	8000	9000	11000	0	8	9	11	0
659.609998	15000	21000	32000	30000	8000	9000	11000	0	8	9	11	0
660.613285	15000	1000	32000	30000	8000	0	11000	0	8	1	11	0
661.616637	15000	21000	1000	30000	8000	9000	0	0	8	8	1	0
662.619980	1000	21000	33000	30000	0	9000	11000	0	1	9	10	0
663.623327	14000	21000	33000	30000	8000	9000	11000	0	7	9	11	0
664.626627	14000	21000	33000	30000	8000	9000	11000	0	8	9	11	0
665.630000	14000	21000	33000	30000	8000	9000	11000	0	8	9	11	0
666.633278	14000	21000	33000	30000	8000	9000	11000	0	8	9	11	0
667.636636	14000	1000	1000	30000	8000	0	0	0	8	1	1	0
668.640342	1000	20000	34000	30000	0	9000	11000	0	1	8	10	0
669.643309	11000	20000	30000	30000	8000	9000	11000	0	7	9	11	0

670.646659	12000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
671.649998	14000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
672.653286	15000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
673.656664	17000	1000	1000	30000	9000	0	0	0	8	1	1	0
674.659954	1000	20000	32000	30000	0	9000	11000	0	1	8	10	0
675.663343	11000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
676.666645	12000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
677.669994	14000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
678.673286	15000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
679.676634	17000	1000	1000	30000	9000	0	0	0	8	1	1	0
680.680004	1000	20000	32000	30000	0	9000	11000	0	1	8	10	0
681.683325	11000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
682.686644	12000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
683.689985	14000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
684.693330	15000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
685.696667	17000	1000	30000	30000	9000	0	11000	0	8	1	11	0
686.699953	1000	21000	1000	30000	0	9000	0	0	1	8	1	0
687.703308	19000	20000	30000	30000	9000	9000	11000	0	8	9	10	0
688.706655	11000	20000	30000	30000	8000	9000	11000	0	9	9	11	0
689.710010	13000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
690.713328	14000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
691.716667	16000	1000	30000	30000	8000	0	11000	0	8	1	11	0
692.719955	1000	21000	1000	30000	0	9000	0	0	1	8	1	0
693.723332	18000	20000	30000	30000	9000	9000	11000	0	7	9	10	0
694.726643	11000	20000	30000	30000	8000	9000	11000	0	9	9	11	0
695.730016	13000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
696.733292	14000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
697.736686	16000	1000	30000	30000	8000	0	11000	0	8	1	11	0
698.739953	1000	21000	1000	30000	0	9000	0	0	1	8	1	0
699.743319	18000	20000	30000	30000	9000	9000	11000	0	7	9	10	0
700.746632	11000	20000	30000	30000	8000	9000	11000	0	9	9	11	0
701.749986	13000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
702.753322	14000	20000	30000	30000	8000	9000	11000	0	8	9	11	0
703.756654	16000	1000	30000	30000	8000	0	11000	0	8	1	11	0
704.759949	16000	21000	1000	30000	8000	9000	0	0	8	8	1	0
705.763304	1000	21000	31000	30000	0	9000	11000	0	1	9	10	0
706.766629	15000	21000	31000	30000	8000	9000	11000	0	7	9	11	0
707.769990	15000	21000	31000	30000	8000	9000	11000	0	8	9	11	0
708.773326	15000	21000	31000	30000	8000	9000	11000	0	8	9	11	0
709.776661	15000	1000	31000	30000	8000	0	11000	0	8	1	11	0
710.779947	15000	22000	1000	30000	8000	9000	0	0	8	8	1	0
711.783298	1000	22000	31000	30000	0	9000	11000	0	1	9	10	0
712.786617	16000	22000	31000	30000	9000	9000	11000	0	7	9	11	0
713.789950	16000	22000	31000	30000	8000	9000	11000	0	9	9	11	0
714.793283	16000	22000	31000	30000	8000	9000	11000	0	8	9	11	0
715.796655	11000	1000	30000	30000	8000	0	11000	0	8	1	11	0
716.800212	11000	21000	1000	30000	8000	9000	0	0	8	8	1	0
717.803713	1000	21000	31000	30000	0	9000	13000	0	1	9	12	0

718.806632	12000	21000	0	30000	8000	9000	0	0	7	9	0	0
719.809963	15000	21000	1000	30000	8000	9000	0	0	8	9	1	0
720.813319	15000	21000	31000	30000	8000	9000	11000	0	8	9	10	0
721.816652	15000	1000	31000	30000	8000	0	11000	0	8	1	11	0
722.819990	15000	22000	31000	30000	8000	9000	11000	0	8	8	11	0
723.823321	1000	22000	31000	30000	0	9000	11000	0	1	9	11	0
724.826611	15000	22000	31000	30000	8000	9000	11000	0	7	9	11	0
725.829989	15000	22000	1000	0	8000	9000	0	0	8	9	1	0
726.833293	15000	22000	33000	0	8000	9000	11000	0	8	9	10	0
727.836607	17000	22000	21000	0	9000	9000	0	0	8	9	11	0
728.839945	20000	1000	0	0	9000	0	0	0	9	1	0	0
729.843315	1000	9000	0	0	0	0	0	0	1	8	0	0
730.846644	16000	0	0	0	0	0	0	0	8	0	0	0
731.849982	0	0	0	0	0	0	0	0	0	0	0	0
732.853319	0	0	0	0	0	0	0	0	0	0	0	0
733.856649	0	0	0	0	0	0	0	0	0	0	0	0
734.859998	0	0	0	0	0	0	0	0	0	0	0	0
735.863317	0	0	0	0	0	0	0	0	0	0	0	0
736.866613	0	0	0	0	0	0	0	0	0	0	0	0
737.869984	0	0	0	0	0	0	0	0	0	0	0	0
738.873311	0	0	0	0	0	0	0	0	0	0	0	0
739.876666	0	0	0	0	0	0	0	0	0	0	0	0
740.879965	0	0	0	0	0	0	0	0	0	0	0	0
741.883322	0	0	0	0	0	0	0	0	0	0	0	0
742.886638	0	0	0	0	0	0	0	0	0	0	0	0
743.890008	0	0	0	0	0	0	0	0	0	0	0	0
744.893280	0	0	0	0	0	0	0	0	0	0	0	0
745.896654	0	0	0	0	0	0	0	0	0	0	0	0
746.899982	0	0	0	0	0	0	0	0	0	0	0	0
747.903321	0	0	0	0	0	0	0	0	0	0	0	0

Table A.2: Client Sending Data - Server Incoming Bandwidth Limited

	Bandwidth (bytes per second)				Bytes Dropped				Retransmitted Packets			
Time Stamp	Group 0	Group 1	Group 2	Group 3	Group 0	Group 1	Group 2	Group 3	Group 0	Group 1	Group 2	Group 3
79.273420	0	0	0	0	0	0	0	0	0	0	0	0
80.276806	0	0	0	0	0	0	0	0	0	0	0	0
81.280048	0	0	0	0	0	0	0	0	0	0	0	0
82.283379	0	0	0	0	0	0	0	0	0	0	0	0
83.286687	0	0	0	0	0	0	0	0	0	0	0	0
84.290047	0	0	0	0	0	0	0	0	0	0	0	0
85.293387	0	0	0	0	0	0	0	0	0	0	0	0
86.296712	0	0	0	0	0	0	0	0	0	0	0	0
87.300048	0	0	0	0	0	0	0	0	0	0	0	0
88.303377	0	0	0	0	0	0	0	0	0	0	0	0
89.306685	0	0	0	0	0	0	0	0	0	0	0	0
90.310041	0	0	0	0	0	0	0	0	0	0	0	0
91.313379	0	0	0	0	0	0	0	0	0	0	0	0
92.316715	0	0	0	0	0	0	0	0	0	0	0	0
93.320043	0	0	0	0	0	0	0	0	0	0	0	0
94.323351	0	0	0	0	0	0	0	0	0	0	0	0
95.326712	0	0	0	0	0	0	0	0	0	0	0	0
96.330043	0	0	0	0	0	0	0	0	0	0	0	0
97.333364	0	0	0	0	0	0	0	0	0	0	0	0
98.336691	0	0	0	0	0	0	0	0	0	0	0	0
99.340047	0	0	0	0	0	0	0	0	0	0	0	0
100.343366	0	0	0	0	0	0	0	0	0	0	0	0
101.346712	0	0	0	0	0	0	0	0	0	0	0	0
102.350045	0	0	0	0	0	0	0	0	0	0	0	0
103.353379	0	0	0	0	0	0	0	0	0	0	0	0
104.356715	0	0	0	0	0	0	0	0	0	0	0	0
105.360009	0	0	0	0	0	0	0	0	0	0	0	0
106.363342	0	0	0	0	0	0	0	0	0	0	0	0
107.366710	0	0	0	0	0	0	0	0	0	0	0	0
108.370000	0	0	0	0	0	0	0	0	0	0	0	0
109.373379	0	0	0	0	0	0	0	0	0	0	0	0
110.376697	0	0	0	0	0	0	0	0	0	0	0	0
111.380042	0	0	0	0	0	0	0	0	0	0	0	0
112.383379	0	0	0	0	0	0	0	0	0	0	0	0
113.386669	0	0	0	0	0	0	0	0	0	0	0	0
114.390045	0	0	0	0	0	0	0	0	0	0	0	0
115.393377	0	0	0	0	0	0	0	0	0	0	0	0
116.396705	0	0	0	0	0	0	0	0	0	0	0	0
117.400004	0	0	0	0	0	0	0	0	0	0	0	0
118.403336	0	0	0	0	0	0	0	0	0	0	0	0

119.406707	0	0	0	0	0	0	0	0	0	0	0	0
120.409999	0	0	0	0	0	0	0	0	0	0	0	0
121.413386	0	0	0	0	0	0	0	0	0	0	0	0
122.416827	0	0	0	0	0	0	0	0	0	0	0	0
123.420045	0	0	0	0	0	0	0	0	0	0	0	0
124.423374	0	0	0	0	0	0	0	0	0	0	0	0
125.426710	0	0	0	0	0	0	0	0	0	0	0	0
126.430019	0	0	0	0	0	0	0	0	0	0	0	0
127.433375	0	0	0	0	0	0	0	0	0	0	0	0
128.436711	0	0	0	0	0	0	0	0	0	0	0	0
129.440050	0	0	0	0	0	0	0	0	0	0	0	0
130.443383	0	0	0	0	0	0	0	0	0	0	0	0
131.446708	0	0	0	0	0	0	0	0	0	0	0	0
132.450021	0	0	0	0	0	0	0	0	0	0	0	0
133.453375	0	0	0	0	0	0	0	0	0	0	0	0
134.456667	0	0	0	0	0	0	0	0	0	0	0	0
135.460051	0	0	0	0	0	0	0	0	0	0	0	0
136.463368	0	0	0	0	0	0	0	0	0	0	0	0
137.466724	0	0	0	0	0	0	0	0	0	0	0	0
138.470002	0	0	0	0	0	0	0	0	0	0	0	0
139.473373	0	0	0	0	0	0	0	0	0	0	0	0
140.476713	0	0	0	0	0	0	0	0	0	0	0	0
141.480045	0	0	0	0	0	0	0	0	0	0	0	0
142.483385	0	0	0	0	0	0	0	0	0	0	0	0
143.486717	0	0	0	0	0	0	0	0	0	0	0	0
144.490020	0	0	0	0	0	0	0	0	0	0	0	0
145.493409	0	0	0	0	0	0	0	0	0	0	0	0
146.496710	0	0	0	0	0	0	0	0	0	0	0	0
147.500053	0	0	0	0	0	0	0	0	0	0	0	0
148.503378	0	0	0	0	0	0	0	0	0	0	0	0
149.506719	0	0	0	0	0	0	0	0	0	0	0	0
150.510025	0	0	0	0	0	0	0	0	0	0	0	0
151.513352	10000	10000	10000	10000	0	0	0	0	0	0	0	0
152.516670	10000	10000	10000	10000	0	0	0	0	0	0	0	0
153.520023	10000	10000	10000	10000	0	0	0	0	0	0	0	0
154.523380	10000	10000	10000	10000	0	0	0	0	0	0	0	0
155.526704	10000	10000	10000	10000	0	0	0	0	0	0	0	0
156.530057	10000	10000	10000	10000	0	0	0	0	0	0	0	0
157.533345	10000	10000	10000	10000	0	0	0	0	0	0	0	0
158.536708	10000	10000	10000	10000	0	0	0	0	0	0	0	0
159.540053	10000	10000	10000	10000	0	0	0	0	0	0	0	0
160.543366	10000	10000	10000	10000	0	0	0	0	0	0	0	0
161.546686	10000	10000	10000	10000	0	0	0	0	0	0	0	0
162.550038	10000	10000	10000	10000	0	0	0	0	0	0	0	0
163.553371	10000	10000	10000	10000	0	0	0	0	0	0	0	0
164.556676	10000	10000	10000	10000	0	0	0	0	0	0	0	0
165.560038	10000	10000	10000	10000	0	0	0	0	0	0	0	0
166.563372	10000	10000	10000	10000	0	0	0	0	0	0	0	0

167.566702	10000	10000	10000	10000	0	0	0	0	0	0	0	0
168.569998	10000	10000	10000	10000	0	0	0	0	0	0	0	0
169.573377	10000	10000	10000	10000	0	0	0	0	0	0	0	0
170.576706	10000	10000	10000	10000	0	0	0	0	0	0	0	0
171.580042	10000	10000	10000	10000	0	0	0	0	0	0	0	0
172.583365	10000	10000	10000	10000	0	0	0	0	0	0	0	0
173.586703	10000	10000	10000	10000	0	0	0	0	0	0	0	0
174.590020	10000	10000	10000	10000	0	0	0	0	0	0	0	0
175.593375	10000	10000	10000	10000	0	0	0	0	0	0	0	0
176.596700	10000	10000	10000	10000	0	0	0	0	0	0	0	0
177.600056	10000	10000	10000	10000	0	0	0	0	0	0	0	0
178.603370	10000	10000	10000	10000	0	0	0	0	0	0	0	0
179.606711	10000	10000	10000	10000	0	0	0	0	0	0	0	0
180.610041	10000	10000	10000	10000	0	0	0	0	0	0	0	0
181.613378	15000	10000	10000	10000	0	0	0	0	0	0	0	0
182.616672	15000	10000	10000	10000	0	0	0	0	0	0	0	0
183.620051	15000	10000	10000	10000	0	0	0	0	0	0	0	0
184.623389	15000	10000	10000	10000	0	0	0	0	0	0	0	0
185.626705	15000	10000	10000	10000	0	0	0	0	0	0	0	0
186.630041	15000	10000	10000	10000	0	0	0	0	0	0	0	0
187.633390	15000	10000	10000	10000	0	0	0	0	0	0	0	0
188.636696	15000	10000	10000	10000	0	0	0	0	0	0	0	0
189.640012	15000	10000	10000	10000	0	0	0	0	0	0	0	0
190.643381	15000	10000	10000	10000	0	0	0	0	0	0	0	0
191.646714	15000	10000	10000	10000	0	0	0	0	0	0	0	0
192.650019	15000	10000	10000	10000	0	0	0	0	0	0	0	0
193.653346	15000	10000	10000	10000	0	0	0	0	0	0	0	0
194.656684	15000	10000	10000	10000	0	0	0	0	0	0	0	0
195.660076	15000	10000	10000	10000	0	0	0	0	0	0	0	0
196.663401	15000	10000	10000	10000	0	0	0	0	0	0	0	0
197.666736	15000	10000	10000	10000	0	0	0	0	0	0	0	0
198.670040	15000	10000	10000	10000	0	0	0	0	0	0	0	0
199.673381	15000	10000	10000	10000	0	0	0	0	0	0	0	0
200.676711	15000	10000	10000	10000	0	0	0	0	0	0	0	0
201.680053	15000	10000	10000	10000	0	0	0	0	0	0	0	0
202.683388	15000	10000	10000	10000	0	0	0	0	0	0	0	0
203.686719	15000	10000	10000	10000	0	0	0	0	0	0	0	0
204.690049	15000	10000	10000	10000	0	0	0	0	0	0	0	0
205.693351	15000	10000	10000	10000	0	0	0	0	0	0	0	0
206.696740	15000	10000	10000	10000	0	0	0	0	0	0	0	0
207.700045	15000	10000	10000	10000	0	0	0	0	0	0	0	0
208.703354	15000	10000	10000	10000	0	0	0	0	0	0	0	0
209.706695	15000	10000	10000	10000	0	0	0	0	0	0	0	0
210.710012	15000	10000	10000	10000	0	0	0	0	0	0	0	0
211.713374	20000	10000	10000	10000	0	0	0	0	0	0	0	0
212.716700	20000	10000	10000	10000	0	0	0	0	0	0	0	0
213.720001	20000	10000	10000	10000	0	0	0	0	0	0	0	0
214.723399	20000	10000	10000	10000	0	0	0	0	0	0	0	0

215.726715	20000	10000	10000	10000	0	0	0	0	0	0	0	0
216.730058	20000	10000	10000	10000	0	0	0	0	0	0	0	0
217.733363	20000	10000	10000	10000	0	0	0	0	0	0	0	0
218.736718	20000	10000	10000	10000	0	0	0	0	0	0	0	0
219.740059	20000	10000	10000	10000	0	0	0	0	0	0	0	0
220.743360	20000	10000	10000	10000	0	0	0	0	0	0	0	0
221.746699	20000	10000	10000	10000	0	0	0	0	0	0	0	0
222.750035	20000	10000	10000	10000	0	0	0	0	0	0	0	0
223.753365	20000	10000	10000	10000	0	0	0	0	0	0	0	0
224.756705	20000	10000	10000	10000	0	0	0	0	0	0	0	0
225.760037	20000	10000	10000	10000	0	0	0	0	0	0	0	0
226.763363	20000	10000	10000	10000	0	0	0	0	0	0	0	0
227.766704	20000	10000	10000	10000	0	0	0	0	0	0	0	0
228.770035	20000	10000	10000	10000	0	0	0	0	0	0	0	0
229.773373	20000	10000	10000	10000	0	0	0	0	0	0	0	0
230.776701	20000	10000	10000	10000	0	0	0	0	0	0	0	0
231.780034	20000	10000	10000	10000	0	0	0	0	0	0	0	0
232.783369	20000	10000	10000	10000	0	0	0	0	0	0	0	0
233.786702	20000	10000	10000	10000	0	0	0	0	0	0	0	0
234.790029	20000	10000	10000	10000	0	0	0	0	0	0	0	0
235.793370	20000	10000	10000	10000	0	0	0	0	0	0	0	0
236.796699	20000	10000	10000	10000	0	0	0	0	0	0	0	0
237.800032	20000	10000	10000	10000	0	0	0	0	0	0	0	0
238.803367	20000	10000	10000	10000	0	0	0	0	0	0	0	0
239.806695	20000	10000	10000	10000	0	0	0	0	0	0	0	0
240.809992	20000	10000	10000	10000	0	0	0	0	0	0	0	0
241.813368	25000	10000	10000	10000	0	0	0	0	0	0	0	0
242.816702	25000	10000	10000	10000	0	0	0	0	0	0	0	0
243.820032	25000	10000	10000	10000	0	0	0	0	0	0	0	0
244.823363	25000	10000	10000	10000	0	0	0	0	0	0	0	0
245.826685	25000	10000	10000	10000	0	0	0	0	0	0	0	0
246.830031	25000	10000	10000	10000	0	0	0	0	0	0	0	0
247.833365	25000	10000	10000	10000	0	0	0	0	0	0	0	0
248.836709	25000	10000	10000	10000	0	0	0	0	0	0	0	0
249.840034	25000	10000	10000	10000	0	0	0	0	0	0	0	0
250.843369	25000	10000	10000	10000	0	0	0	0	0	0	0	0
251.846671	25000	10000	10000	10000	0	0	0	0	0	0	0	0
252.850032	25000	10000	10000	10000	0	0	0	0	0	0	0	0
253.853377	25000	10000	10000	10000	0	0	0	0	0	0	0	0
254.856671	25000	10000	10000	10000	0	0	0	0	0	0	0	0
255.860024	25000	10000	10000	10000	0	0	0	0	0	0	0	0
256.863367	25000	10000	10000	10000	0	0	0	0	0	0	0	0
257.866669	25000	10000	10000	10000	0	0	0	0	0	0	0	0
258.870039	25000	10000	10000	10000	0	0	0	0	0	0	0	0
259.873372	25000	10000	10000	10000	0	0	0	0	0	0	0	0
260.876689	25000	10000	10000	10000	0	0	0	0	0	0	0	0
261.880035	25000	10000	10000	10000	0	0	0	0	0	0	0	0
262.883364	25000	10000	10000	10000	0	0	0	0	0	0	0	0

263.886703	25000	10000	10000	10000	0	0	0	0	0	0	0	0
264.890040	25000	10000	10000	10000	0	0	0	0	0	0	0	0
265.893369	25000	10000	10000	10000	0	0	0	0	0	0	0	0
266.896689	25000	10000	10000	10000	0	0	0	0	0	0	0	0
267.900058	25000	10000	10000	10000	0	0	0	0	0	0	0	0
268.903361	25000	10000	10000	10000	0	0	0	0	0	0	0	0
269.906711	25000	10000	10000	10000	0	0	0	0	0	0	0	0
270.910032	25000	10000	10000	10000	0	0	0	0	0	0	0	0
271.913350	30000	10000	10000	10000	0	0	0	0	0	0	0	0
272.916704	30000	10000	10000	10000	0	0	0	0	0	0	0	0
273.920069	30000	10000	10000	10000	0	0	0	0	0	0	0	0
274.923364	30000	10000	10000	10000	0	0	0	0	0	0	0	0
275.926708	30000	10000	10000	10000	0	0	0	0	0	0	0	0
276.930063	30000	10000	10000	10000	0	0	0	0	0	0	0	0
277.933380	30000	10000	10000	10000	0	0	0	0	0	0	0	0
278.936714	30000	10000	10000	10000	0	0	0	0	0	0	0	0
279.940012	30000	10000	10000	10000	0	0	0	0	0	0	0	0
280.943355	30000	10000	10000	10000	0	0	0	0	0	0	0	0
281.946693	30000	10000	10000	10000	0	0	0	0	0	0	0	0
282.950028	30000	10000	10000	10000	0	0	0	0	0	0	0	0
283.953359	30000	10000	10000	10000	0	0	0	0	0	0	0	0
284.956691	30000	10000	10000	10000	0	0	0	0	0	0	0	0
285.959999	30000	10000	10000	10000	0	0	0	0	0	0	0	0
286.963358	30000	10000	10000	10000	0	0	0	0	0	0	0	0
287.966699	30000	10000	10000	10000	0	0	0	0	0	0	0	0
288.970029	30000	10000	10000	10000	0	0	0	0	0	0	0	0
289.973361	30000	10000	10000	10000	0	0	0	0	0	0	0	0
290.976693	30000	10000	10000	10000	0	0	0	0	0	0	0	0
291.980000	30000	10000	10000	10000	0	0	0	0	0	0	0	0
292.983366	30000	10000	10000	10000	0	0	0	0	0	0	0	0
293.986699	30000	10000	10000	10000	0	0	0	0	0	0	0	0
294.990033	30000	10000	10000	10000	0	0	0	0	0	0	0	0
295.993366	30000	10000	10000	10000	0	0	0	0	0	0	0	0
296.996697	30000	10000	10000	10000	0	0	0	0	0	0	0	0
298.000004	30000	10000	10000	10000	0	0	0	0	0	0	0	0
299.003328	30000	10000	10000	10000	0	0	0	0	0	0	0	0
300.006692	30000	10000	10000	10000	0	0	0	0	0	0	0	0
301.010027	30000	10000	10000	10000	0	0	0	0	0	0	0	0
302.013360	30000	15000	10000	10000	0	0	0	0	0	0	0	0
303.016680	30000	15000	10000	10000	0	0	0	0	0	0	0	0
304.020040	30000	15000	10000	10000	0	0	0	0	0	0	0	0
305.023364	30000	15000	10000	10000	0	0	0	0	0	0	0	0
306.026690	30000	15000	10000	10000	0	0	0	0	0	0	0	0
307.030041	30000	15000	10000	10000	0	0	0	0	0	0	0	0
308.033356	30000	15000	10000	10000	0	0	0	0	0	0	0	0
309.036662	30000	15000	10000	10000	0	0	0	0	0	0	0	0
310.040032	30000	15000	10000	10000	0	0	0	0	0	0	0	0
311.043328	30000	15000	10000	10000	0	0	0	0	0	0	0	0

312.046690	30000	15000	10000	10000	0	0	0	0	0	0	0	0
313.050030	30000	15000	10000	10000	0	0	0	0	0	0	0	0
314.053357	30000	15000	10000	10000	0	0	0	0	0	0	0	0
315.056708	30000	15000	10000	10000	0	0	0	0	0	0	0	0
316.060051	30000	15000	10000	10000	0	0	0	0	0	0	0	0
317.063362	30000	15000	10000	10000	0	0	0	0	0	0	0	0
318.066692	30000	15000	10000	10000	0	0	0	0	0	0	0	0
319.070029	30000	15000	10000	10000	0	0	0	0	0	0	0	0
320.073320	30000	15000	10000	10000	0	0	0	0	0	0	0	0
321.076689	30000	15000	10000	10000	0	0	0	0	0	0	0	0
322.080010	30000	15000	10000	10000	0	0	0	0	0	0	0	0
323.083330	30000	15000	10000	10000	0	0	0	0	0	0	0	0
324.086714	30000	15000	10000	10000	0	0	0	0	0	0	0	0
325.090046	30000	15000	10000	10000	0	0	0	0	0	0	0	0
326.093371	30000	15000	10000	10000	0	0	0	0	0	0	0	0
327.096694	30000	15000	10000	10000	0	0	0	0	0	0	0	0
328.100022	30000	15000	10000	10000	0	0	0	0	0	0	0	0
329.103362	30000	15000	10000	10000	0	0	0	0	0	0	0	0
330.106700	30000	15000	10000	10000	0	0	0	0	0	0	0	0
331.110046	30000	15000	10000	10000	0	0	0	0	0	0	0	0
332.113356	30000	20000	10000	10000	0	0	0	0	0	0	0	0
333.116695	30000	20000	10000	10000	0	0	0	0	0	0	0	0
334.120045	30000	20000	10000	10000	0	0	0	0	0	0	0	0
335.123363	30000	20000	10000	10000	0	0	0	0	0	0	0	0
336.126688	30000	20000	10000	10000	0	0	0	0	0	0	0	0
337.130005	30000	20000	10000	10000	0	0	0	0	0	0	0	0
338.133357	30000	20000	10000	10000	0	0	0	0	0	0	0	0
339.136715	30000	20000	10000	10000	0	0	0	0	0	0	0	0
340.140040	30000	20000	10000	10000	0	0	0	0	0	0	0	0
341.143313	30000	20000	10000	10000	0	0	0	0	0	0	0	0
342.146687	30000	20000	10000	10000	0	0	0	0	0	0	0	0
343.150022	30000	20000	10000	10000	0	0	0	0	0	0	0	0
344.153358	30000	20000	10000	10000	0	0	0	0	0	0	0	0
345.156690	30000	20000	10000	10000	0	0	0	0	0	0	0	0
346.160031	30000	20000	10000	10000	0	0	0	0	0	0	0	0
347.163328	30000	20000	10000	10000	0	0	0	0	0	0	0	0
348.166690	30000	20000	10000	10000	0	0	0	0	0	0	0	0
349.170020	30000	20000	10000	10000	0	0	0	0	0	0	0	0
350.173357	30000	20000	10000	10000	0	0	0	0	0	0	0	0
351.176681	30000	20000	10000	10000	0	0	0	0	0	0	0	0
352.180020	30000	20000	10000	10000	0	0	0	0	0	0	0	0
353.183358	30000	20000	10000	10000	0	0	0	0	0	0	0	0
354.187169	30000	20000	10000	10000	0	0	0	0	0	0	0	0
355.190024	30000	20000	10000	10000	0	0	0	0	0	0	0	0
356.193354	30000	20000	10000	10000	0	0	0	0	0	0	0	0
357.196698	30000	20000	10000	10000	0	0	0	0	0	0	0	0
358.200030	30000	20000	10000	10000	0	0	0	0	0	0	0	0
359.203356	30000	20000	10000	10000	0	0	0	0	0	0	0	0

360.206687	30000	20000	10000	10000	0	0	0	0	0	0	0	0
361.210021	30000	20000	10000	10000	0	0	0	0	0	0	0	0
362.213344	30000	25000	10000	10000	0	0	0	0	0	0	0	0
363.216684	30000	25000	10000	10000	0	0	0	0	0	0	0	0
364.220022	30000	25000	10000	10000	0	0	0	0	0	0	0	0
365.223361	30000	25000	10000	10000	0	0	0	0	0	0	0	0
366.226682	30000	25000	10000	10000	0	0	0	0	0	0	0	0
367.230049	30000	25000	10000	10000	0	0	0	0	0	0	0	0
368.233368	30000	25000	10000	10000	0	0	0	0	0	0	0	0
369.236686	30000	25000	10000	10000	0	0	0	0	0	0	0	0
370.240011	30000	25000	10000	10000	0	0	0	0	0	0	0	0
371.243577	30000	25000	10000	10000	0	0	0	0	0	0	0	0
372.246691	30000	25000	10000	10000	0	0	0	0	0	0	0	0
373.250008	30000	25000	10000	10000	0	0	0	0	0	0	0	0
374.253369	30000	25000	10000	10000	0	0	0	0	0	0	0	0
375.256692	30000	25000	10000	10000	0	0	0	0	0	0	0	0
376.260020	30000	25000	10000	10000	0	0	0	0	0	0	0	0
377.263348	30000	25000	10000	10000	0	0	0	0	0	0	0	0
378.266702	30000	25000	10000	10000	0	0	0	0	0	0	0	0
379.270022	30000	25000	10000	10000	0	0	0	0	0	0	0	0
380.273356	30000	25000	10000	10000	0	0	0	0	0	0	0	0
381.276715	30000	25000	10000	10000	0	0	0	0	0	0	0	0
382.280004	30000	25000	10000	10000	0	0	0	0	0	0	0	0
383.283369	30000	25000	10000	10000	0	0	0	0	0	0	0	0
384.286665	30000	25000	10000	10000	0	0	0	0	0	0	0	0
385.290029	30000	25000	10000	10000	0	0	0	0	0	0	0	0
386.293385	30000	25000	10000	10000	0	0	0	0	0	0	0	0
387.296687	30000	25000	10000	10000	0	0	0	0	0	0	0	0
388.300017	30000	25000	10000	10000	0	0	0	0	0	0	0	0
389.303353	30000	25000	10000	10000	0	0	0	0	0	0	0	0
390.306687	30000	25000	10000	10000	0	0	0	0	0	0	0	0
391.310024	30000	25000	10000	10000	0	0	0	0	0	0	0	0
392.313349	30000	30000	10000	10000	0	0	0	0	0	0	0	0
393.316688	30000	30000	10000	10000	0	0	0	0	0	0	0	0
394.320014	30000	30000	10000	10000	0	0	0	0	0	0	0	0
395.323631	30000	30000	10000	11000	0	0	0	0	0	0	0	0
396.326721	30000	30000	12000	18000	0	0	0	0	0	0	0	0
397.330199	30000	31000	17000	10000	0	0	0	0	0	0	0	0
398.333519	30000	42000	10000	10000	0	0	0	0	0	0	0	0
399.336822	30000	39000	10000	10000	0	0	0	0	0	0	0	0
400.340301	36000	37000	10000	10000	0	0	0	0	0	0	0	0
401.343381	39000	20000	10000	10000	0	0	0	0	0	0	0	0
402.346815	33000	40000	10000	10000	0	0	0	0	0	0	0	0
403.350036	39000	30000	10000	10000	0	0	0	0	0	0	0	0
404.353306	32000	30000	10000	10000	0	0	0	0	0	0	0	0
405.356642	30000	30000	10000	10000	0	0	0	0	0	0	0	0
406.359974	30000	30000	10000	10000	0	0	0	0	0	0	0	0
407.363306	30000	30000	10000	10000	0	0	0	0	0	0	0	0

408.366639	30000	30000	10000	10000	0	0	0	0	0	0	0	0
409.369976	30000	30000	10000	10000	0	0	0	0	0	0	0	0
410.373306	30000	30000	10000	10000	0	0	0	0	0	0	0	0
411.376641	30000	30000	10000	10000	0	0	0	0	0	0	0	0
412.379980	30000	30000	10000	10000	0	0	0	0	0	0	0	0
413.383309	30000	30000	10000	10000	0	0	0	0	0	0	0	0
414.386647	31000	31000	10000	10000	0	0	0	0	0	0	0	0
415.389977	29000	29000	10000	10000	0	0	0	0	0	0	0	0
416.393306	30000	31000	10000	10000	0	0	0	0	0	0	0	0
417.396641	30000	29000	10000	10000	0	0	0	0	0	0	0	0
418.399975	30000	30000	10000	10000	0	0	0	0	0	0	0	0
419.403313	30000	30000	10000	10000	0	0	0	0	0	0	0	0
420.406655	30000	30000	10000	10000	0	0	0	0	0	0	0	0
421.409977	31000	31000	16000	10000	0	0	0	0	0	0	0	0
422.413336	29000	29000	15000	10000	0	0	0	0	0	0	0	0
423.416691	31000	30000	15000	10000	0	0	0	0	0	0	0	0
424.419977	29000	31000	15000	10000	0	0	0	0	0	0	0	0
425.423339	30000	29000	15000	10000	0	0	0	0	0	0	0	0
426.426688	31000	30000	15000	10000	0	0	0	0	0	0	0	0
427.429990	29000	30000	15000	10000	0	0	0	0	0	0	0	0
428.433321	30000	30000	15000	10000	0	0	0	0	0	0	0	0
429.436641	30000	30000	15000	10000	0	0	0	0	0	0	0	0
430.439991	30000	30000	15000	10000	0	0	0	0	0	0	0	0
431.443307	30000	31000	15000	10000	0	0	0	0	0	0	0	0
432.446659	30000	29000	15000	10000	0	0	0	0	0	0	0	0
433.449999	30000	30000	15000	10000	0	0	0	0	0	0	0	0
434.453323	30000	30000	15000	10000	0	0	0	0	0	0	0	0
435.456688	30000	30000	15000	10000	0	0	0	0	0	0	0	0
436.460017	30000	30000	15000	10000	0	0	0	0	0	0	0	0
437.463337	30000	30000	15000	10000	0	0	0	0	0	0	0	0
438.466659	30000	30000	15000	10000	0	0	0	0	0	0	0	0
439.469991	30000	30000	15000	10000	0	0	0	0	0	0	0	0
440.473320	30000	30000	15000	10000	0	0	0	0	0	0	0	0
441.476688	30000	30000	15000	10000	0	0	0	0	0	0	0	0
442.480008	30000	30000	15000	10000	0	0	0	0	0	0	0	0
443.483365	30000	30000	15000	10000	0	0	0	0	0	0	0	0
444.486674	30000	30000	15000	10000	0	0	0	0	0	0	0	0
445.490001	30000	31000	15000	10000	0	0	0	0	0	0	0	0
446.493322	30000	29000	15000	10000	0	0	0	0	0	0	0	0
447.496668	30000	30000	15000	10000	0	0	0	0	0	0	0	0
448.500016	30000	31000	15000	10000	0	0	0	0	0	0	0	0
449.503375	30000	30000	15000	10000	0	0	0	0	0	0	0	0
450.506652	30000	29000	14000	10000	0	0	0	0	0	0	0	0
451.510018	31000	30000	20000	10000	0	0	0	0	0	0	0	0
452.513328	30000	30000	20000	10000	0	0	0	0	0	0	0	0
453.516695	29000	31000	20000	10000	0	0	0	0	0	0	0	0
454.519995	30000	29000	20000	10000	0	0	0	0	0	0	0	0
455.523324	30000	30000	20000	10000	0	0	0	0	0	0	0	0

456.526668	31000	30000	20000	10000	0	0	0	0	0	0	0	0
457.530009	29000	30000	20000	10000	0	0	0	0	0	0	0	0
458.533374	30000	30000	20000	10000	0	0	0	0	0	0	0	0
459.536673	30000	30000	20000	11000	0	0	0	0	0	0	0	0
460.540069	30000	30000	20000	10000	0	0	0	0	0	0	0	0
461.543309	30000	30000	20000	10000	0	0	0	0	0	0	0	0
462.546646	30000	30000	20000	10000	0	0	0	0	0	0	0	0
463.549972	30000	31000	21000	10000	0	0	0	0	0	0	0	0
464.553301	30000	30000	20000	10000	0	0	0	0	0	0	0	0
465.556634	30000	30000	20000	10000	0	0	0	0	0	0	0	0
466.559968	31000	30000	20000	10000	0	0	0	0	0	0	0	0
467.563304	30000	30000	20000	10000	0	0	0	0	0	0	0	0
468.566648	30000	30000	20000	10000	0	0	0	0	0	0	0	0
469.569990	30000	30000	20000	10000	0	0	0	0	0	0	0	0
470.573297	30000	30000	20000	10000	0	0	0	0	0	0	0	0
471.576648	30000	30000	20000	10000	0	0	0	0	0	0	0	0
472.580000	30000	30000	20000	10000	0	0	0	0	0	0	0	0
473.583305	30000	30000	20000	10000	0	0	0	0	0	0	0	0
474.586658	30000	30000	20000	10000	0	0	0	0	0	0	0	0
475.589977	30000	30000	20000	10000	0	0	0	0	0	0	0	0
476.593306	30000	30000	20000	10000	0	0	0	0	0	0	0	0
477.596651	30000	30000	20000	10000	0	0	0	0	0	0	0	0
478.600002	30000	30000	20000	10000	0	0	0	0	0	0	0	0
479.603322	30000	30000	20000	10000	0	0	0	0	0	0	0	0
480.606643	30000	30000	20000	10000	0	0	0	0	0	0	0	0
481.610011	30000	30000	25000	10000	0	0	0	0	0	0	0	0
482.613340	30000	30000	25000	10000	0	0	0	0	0	0	0	0
483.616638	30000	30000	25000	10000	0	0	0	0	0	0	0	0
484.619984	30000	30000	25000	10000	0	0	0	0	0	0	0	0
485.623349	30000	30000	25000	10000	0	0	0	0	0	0	0	0
486.626646	30000	30000	25000	10000	0	0	0	0	0	0	0	0
487.630041	30000	30000	25000	10000	0	0	0	0	0	0	0	0
488.633321	30000	30000	25000	10000	0	0	0	0	0	0	0	0
489.636692	30000	30000	25000	10000	0	0	0	0	0	0	0	0
490.640009	30000	30000	25000	10000	0	0	0	0	0	0	0	0
491.643317	30000	30000	25000	10000	0	0	0	0	0	0	0	0
492.646683	30000	30000	25000	10000	0	0	0	0	0	0	0	0
493.649998	30000	30000	25000	10000	0	0	0	0	0	0	0	0
494.653319	30000	30000	25000	10000	0	0	0	0	0	0	0	0
495.656689	30000	30000	25000	10000	0	0	0	0	0	0	0	0
496.660025	30000	30000	25000	10000	0	0	0	0	0	0	0	0
497.663302	30000	30000	25000	10000	0	0	0	0	0	0	0	0
498.666702	30000	30000	25000	10000	0	0	0	0	0	0	0	0
499.670017	30000	30000	25000	10000	0	0	0	0	0	0	0	0
500.673359	30000	30000	25000	10000	0	0	0	0	0	0	0	0
501.676679	30000	30000	25000	10000	0	0	0	0	0	0	0	0
502.679986	30000	30000	25000	10000	0	0	0	0	0	0	0	0
503.683362	30000	30000	25000	10000	0	0	0	0	0	0	0	0

504.686642	30000	30000	25000	10000	0	0	0	0	0	0	0	0
505.689964	30000	30000	25000	10000	0	0	0	0	0	0	0	0
506.693357	30000	30000	25000	10000	0	0	0	0	0	0	0	0
507.696660	30000	30000	25000	10000	0	0	0	0	0	0	0	0
508.700032	30000	30000	25000	10000	0	0	0	0	0	0	0	0
509.703325	30000	30000	25000	10000	0	0	0	0	0	0	0	0
510.706701	30000	30000	25000	10000	0	0	0	0	0	0	0	0
511.709976	29000	29000	30000	10000	0	0	0	0	0	0	0	0
512.713337	22000	34000	30000	10000	0	0	0	0	19	9	0	0
513.716698	19000	30000	30000	10000	0	0	0	0	12	13	0	0
514.720000	0	1000	30000	10000	0	0	0	0	0	1	0	0
515.723327	21000	30000	30000	10000	0	0	0	0	7	8	0	0
516.726665	1000	31000	30000	10000	0	0	0	0	1	9	0	0
517.730052	22000	31000	30000	10000	0	0	0	0	7	9	0	0
518.733332	24000	33000	30000	10000	0	0	0	0	8	9	0	0
519.736668	24000	34000	30000	10000	0	0	0	0	8	9	0	0
520.740019	28000	1000	30000	10000	0	0	0	0	8	1	0	0
521.743302	28000	33000	30000	10000	0	0	0	0	9	8	0	0
522.746662	1000	34000	30000	10000	0	0	0	0	1	9	0	0
523.749989	29000	34000	30000	10000	0	0	0	0	8	9	0	0
524.753321	31000	39000	30000	10000	0	0	0	0	9	9	0	0
525.756654	31000	40000	30000	10000	0	0	0	0	9	11	0	0
526.759979	33000	1000	30000	10000	0	0	0	0	9	1	0	0
527.763610	33000	40000	30000	10000	0	0	0	0	9	11	0	0
528.766638	1000	39000	30000	10000	0	0	0	0	1	10	0	0
529.770275	37000	40000	30000	10000	0	0	0	0	9	11	0	0
530.773346	37000	42000	30000	10000	0	0	0	0	10	11	0	0
531.776673	38000	43000	30000	10000	0	0	0	0	11	11	0	0
532.779965	39000	43000	30000	10000	0	0	0	0	11	11	0	0
533.783315	19000	1000	30000	10000	0	0	0	0	11	1	0	0
534.786650	0	28000	30000	10000	0	0	0	0	0	10	0	0
535.789963	20000	31000	30000	10000	0	0	0	0	8	9	0	0
536.793333	1000	33000	30000	10000	0	0	0	0	1	9	0	0
537.796674	19000	33000	30000	10000	0	0	0	0	7	9	0	0
538.799978	20000	37000	30000	10000	0	0	0	0	8	9	0	0
539.803294	23000	1000	30000	10000	0	0	0	0	8	1	0	0
540.806668	23000	36000	30000	10000	0	0	0	0	8	10	0	0
541.809976	26000	37000	30000	15000	0	0	0	0	8	11	0	0
542.813335	1000	40000	30000	15000	0	0	0	0	1	11	0	0
543.816669	25000	40000	30000	15000	0	0	0	0	8	11	0	0
544.819980	26000	42000	30000	15000	0	0	0	0	9	11	0	0
545.823337	28000	1000	30000	15000	0	0	0	0	9	1	0	0
546.826670	28000	41000	30000	15000	0	0	0	0	9	10	0	0
547.829971	30000	42000	30000	15000	0	0	0	0	9	11	0	0
548.833338	1000	42000	30000	15000	0	0	0	0	1	11	0	0
549.836671	30000	43000	30000	15000	0	0	0	0	8	11	0	0
550.840006	31000	42000	30000	15000	0	0	0	0	9	11	0	0
551.843300	32000	42000	30000	15000	0	0	0	0	9	11	0	0

552.846670	31000	1000	30000	15000	0	0	0	0	9	1	0	0
553.849999	31000	41000	30000	15000	0	0	0	0	9	10	0	0
554.853332	1000	42000	30000	15000	0	0	0	0	1	11	0	0
555.856673	30000	44000	30000	15000	0	0	0	0	8	11	0	0
556.859960	31000	30000	30000	15000	0	0	0	0	9	11	0	0
557.863323	20000	30000	30000	15000	0	0	0	0	9	9	0	0
558.866674	21000	1000	30000	15000	0	0	0	0	8	1	0	0
559.870022	22000	29000	30000	15000	0	0	0	0	8	8	0	0
560.873340	1000	30000	30000	15000	0	0	0	0	1	9	0	0
561.876683	23000	32000	30000	15000	0	0	0	0	7	9	0	0
562.879994	24000	32000	30000	15000	0	0	0	0	8	9	0	0
563.883323	28000	32000	30000	15000	0	0	0	0	8	9	0	0
564.886673	29000	1000	30000	15000	0	0	0	0	9	1	0	0
565.890023	30000	31000	30000	15000	0	0	0	0	9	8	0	0
566.893339	1000	32000	30000	15000	0	0	0	0	1	9	0	0
567.896675	31000	37000	30000	15000	0	0	0	0	8	9	0	0
568.900001	32000	37000	30000	15000	0	0	0	0	9	11	0	0
569.903345	34000	37000	30000	15000	0	0	0	0	9	11	0	0
570.906670	37000	1000	30000	15000	0	0	0	0	9	1	0	0
571.910001	19000	36000	30000	28000	0	0	0	0	11	10	0	8
572.913353	0	29000	35000	20000	0	0	0	0	0	11	5	0
573.916672	1000	32000	14000	20000	0	0	0	0	1	9	0	0
574.920005	17000	29000	43000	20000	0	0	0	0	7	9	12	0
575.923327	18000	29000	43000	20000	0	0	0	0	8	9	11	0
576.926682	20000	1000	1000	20000	0	0	0	0	8	1	1	0
577.929999	21000	28000	42000	20000	0	0	0	0	8	8	10	0
578.933345	22000	29000	43000	20000	0	0	0	0	8	9	11	0
579.936662	1000	31000	43000	20000	0	0	0	0	1	9	11	0
580.940026	21000	30000	46000	20000	0	0	0	0	7	9	11	0
581.943318	22000	30000	47000	20000	0	0	0	0	8	9	11	0
582.946661	22000	1000	1000	20000	0	0	0	0	8	1	1	0
583.949995	26000	29000	47000	20000	0	0	0	0	8	8	10	0
584.953329	27000	30000	48000	20000	0	0	0	0	9	9	11	0
585.956663	1000	33000	48000	20000	0	0	0	0	1	9	11	0
586.959993	26000	32000	51000	20000	0	0	0	0	8	9	11	0
587.963331	27000	29000	42000	20000	0	0	0	0	9	9	12	0
588.966628	18000	1000	44000	21000	0	0	0	0	9	1	11	1
589.969981	20000	28000	1000	20000	0	0	0	0	8	8	1	0
590.973329	20000	29000	43000	20000	0	0	0	0	8	9	10	0
591.976621	22000	29000	44000	20000	0	0	0	0	8	9	11	0
592.979979	1000	30000	47000	20000	0	0	0	0	1	9	11	0
593.983332	17000	29000	42000	20000	0	0	0	0	7	9	11	0
594.986645	18000	1000	44000	20000	0	0	0	0	8	1	11	0
595.989982	20000	28000	1000	20000	0	0	0	0	8	8	1	0
596.993338	20000	29000	43000	20000	0	0	0	0	8	9	10	0
597.996626	22000	29000	44000	20000	0	0	0	0	8	9	11	0
598.999981	1000	30000	47000	20000	0	0	0	0	1	9	11	0
600.003300	17000	29000	42000	20000	0	0	0	0	7	9	11	0

601.006624	18000	1000	44000	20000	0	0	0	0	8	1	11	0
602.010000	20000	28000	1000	25000	0	0	0	0	8	8	1	0
603.013328	20000	29000	43000	25000	0	0	0	0	8	9	10	0
604.016665	22000	29000	44000	25000	0	0	0	0	8	9	11	0
605.019990	1000	30000	47000	25000	0	0	0	0	1	9	11	0
606.023309	17000	29000	42000	25000	0	0	0	0	7	9	11	0
607.026621	18000	29000	44000	25000	0	0	0	0	8	9	11	0
608.029997	20000	1000	1000	25000	0	0	0	0	8	1	1	0
609.033330	20000	28000	43000	25000	0	0	0	0	8	8	10	0
610.036664	22000	29000	44000	25000	0	0	0	0	8	9	11	0
611.039979	1000	31000	44000	25000	0	0	0	0	1	9	11	0
612.043283	21000	30000	46000	25000	0	0	0	0	7	9	11	0
613.046619	22000	30000	46000	25000	0	0	0	0	8	9	11	0
614.049992	26000	1000	1000	25000	0	0	0	0	8	1	1	0
615.053306	27000	29000	45000	25000	0	0	0	0	9	8	10	0
616.056654	19000	30000	46000	25000	0	0	0	0	9	9	11	0
617.060002	1000	31000	41000	25000	0	0	0	0	1	9	11	0
618.063301	18000	30000	43000	25000	0	0	0	0	7	9	11	0
619.066644	19000	30000	44000	25000	0	0	0	0	8	9	11	0
620.072837	21000	1000	44000	25000	0	0	0	0	8	1	11	0
621.073645	21000	29000	1000	25000	0	0	0	0	8	8	1	0
622.076662	23000	30000	43000	25000	0	0	0	0	8	9	10	0
623.080001	1000	33000	44000	25000	0	0	0	0	1	9	11	0
624.083330	22000	32000	47000	25000	0	0	0	0	7	9	11	0
625.086648	23000	32000	47000	25000	0	0	0	0	8	9	11	0
626.089954	26000	1000	47000	25000	0	0	0	0	8	1	11	0
627.093306	19000	31000	1000	25000	0	0	0	0	9	8	1	0
628.096666	21000	29000	40000	26000	0	0	0	0	8	9	10	1
629.100018	1000	31000	41000	25000	0	0	0	0	1	9	11	0
630.103332	20000	30000	43000	25000	0	0	0	0	7	9	11	0
631.106660	21000	30000	43000	25000	0	0	0	0	8	9	11	0
632.109952	23000	1000	43000	30000	0	0	0	0	8	1	11	0
633.113332	22000	29000	1000	30000	0	0	0	0	8	8	1	0
634.116658	22000	30000	42000	30000	0	0	0	0	8	9	10	0
635.120019	1000	32000	43000	30000	0	0	0	0	1	9	11	0
636.123285	21000	31000	44000	30000	0	0	0	0	7	9	11	0
637.126662	22000	31000	43000	30000	0	0	0	0	8	9	11	0
638.129975	24000	1000	43000	30000	0	0	0	0	8	1	11	0
639.133325	2000	30000	1000	30000	0	0	0	0	2	8	1	0
640.136641	0	29000	41000	30000	0	0	0	0	0	9	10	0
641.139998	19000	31000	42000	30000	0	0	0	0	8	9	11	0
642.143318	19000	30000	44000	30000	0	0	0	0	8	9	11	0
643.146659	22000	30000	43000	30000	0	0	0	0	8	9	11	0
644.149992	23000	1000	43000	30000	0	0	0	0	8	1	11	0
645.153280	23000	29000	1000	30000	0	0	0	0	8	8	1	0
646.156659	1000	30000	42000	30000	0	0	0	0	1	9	10	0
647.159964	22000	31000	43000	30000	0	0	0	0	7	9	11	0
648.163325	23000	30000	45000	30000	0	0	0	0	8	9	11	0

649.166657	23000	30000	44000	30000	0	0	0	0	8	9	11	0
650.169994	22000	1000	44000	30000	0	0	0	0	8	1	11	0
651.173322	22000	29000	1000	30000	0	0	0	0	8	8	1	0
652.176661	1000	30000	43000	30000	0	0	0	0	1	9	10	0
653.179996	21000	30000	44000	30000	0	0	0	0	7	9	11	0
654.183328	22000	30000	46000	30000	0	0	0	0	8	9	11	0
655.186656	20000	29000	41000	30000	0	0	0	0	8	9	11	0
656.189993	20000	1000	41000	30000	0	0	0	0	8	1	11	0
657.193288	22000	28000	1000	30000	0	0	0	0	8	8	1	0
658.196656	1000	29000	40000	30000	0	0	0	0	1	9	10	0
659.199998	22000	29000	41000	30000	0	0	0	0	7	9	11	0
660.203325	26000	30000	44000	30000	0	0	0	0	8	9	11	0
661.206658	20000	29000	41000	30000	0	0	0	0	9	9	11	0
662.209987	20000	1000	41000	30000	0	0	0	0	8	1	11	0
663.213325	22000	28000	1000	30000	0	0	0	0	8	8	1	0
664.216662	1000	29000	40000	30000	0	0	0	0	1	9	10	0
665.220000	22000	29000	41000	30000	0	0	0	0	7	9	11	0
666.223334	26000	30000	44000	30000	0	0	0	0	8	9	11	0
667.226663	20000	29000	41000	30000	0	0	0	0	9	9	11	0
668.229952	20000	29000	41000	30000	0	0	0	0	8	9	11	0
669.233328	22000	1000	1000	30000	0	0	0	0	8	1	1	0
670.236653	1000	28000	40000	30000	0	0	0	0	1	8	10	0
671.239978	22000	29000	41000	30000	0	0	0	0	7	9	11	0
672.243293	26000	31000	41000	30000	0	0	0	0	8	9	11	0
673.246668	29000	29000	42000	30000	0	0	0	0	9	9	11	0
674.249950	19000	29000	41000	30000	0	0	0	0	9	9	11	0
675.253309	21000	1000	41000	30000	0	0	0	0	8	1	11	0
676.256654	1000	28000	1000	30000	0	0	0	0	1	8	1	0
677.260002	21000	29000	40000	30000	0	0	0	0	7	9	10	0
678.263329	24000	31000	41000	30000	0	0	0	0	8	9	11	0
679.266652	28000	29000	42000	30000	0	0	0	0	8	9	11	0
680.269981	19000	29000	41000	30000	0	0	0	0	9	9	11	0
681.273326	21000	1000	41000	30000	0	0	0	0	8	1	11	0
682.276657	1000	28000	1000	30000	0	0	0	0	1	8	1	0
683.280011	21000	29000	40000	30000	0	0	0	0	7	9	10	0
684.283331	24000	31000	41000	30000	0	0	0	0	8	9	11	0
685.286676	28000	29000	42000	30000	0	0	0	0	8	9	11	0
686.289975	19000	29000	41000	30000	0	0	0	0	9	9	11	0
687.293323	21000	1000	41000	30000	0	0	0	0	8	1	11	0
688.296654	1000	28000	1000	30000	0	0	0	0	1	8	1	0
689.299989	21000	29000	40000	30000	0	0	0	0	7	9	10	0
690.303319	24000	31000	41000	30000	0	0	0	0	8	9	11	0
691.306651	24000	30000	43000	30000	0	0	0	0	8	9	11	0
692.309988	24000	30000	42000	30000	0	0	0	0	8	9	11	0
693.313318	23000	1000	42000	30000	0	0	0	0	8	1	11	0
694.316655	1000	29000	1000	30000	0	0	0	0	1	8	1	0
695.319952	22000	30000	41000	30000	0	0	0	0	7	9	10	0
696.323324	23000	32000	42000	30000	0	0	0	0	8	9	11	0

697.326660	23000	31000	43000	30000	0	0	0	0	8	9	11	0
698.329986	26000	31000	42000	30000	0	0	0	0	8	9	11	0
699.333320	24000	1000	42000	30000	0	0	0	0	9	1	11	0
700.336830	24000	30000	1000	30000	0	0	0	0	8	8	1	0
701.339973	1000	31000	41000	30000	0	0	0	0	1	9	10	0
702.343312	18000	31000	41000	30000	0	0	0	0	7	9	11	0
703.346651	19000	30000	43000	30000	0	0	0	0	8	9	11	0
704.349979	21000	30000	2000	30000	0	0	0	0	8	9	2	0
705.353318	23000	1000	0	30000	0	0	0	0	8	1	0	0
706.356610	23000	29000	43000	30000	0	0	0	0	8	8	11	0
707.359966	1000	30000	42000	30000	0	0	0	0	1	9	11	0
708.363318	22000	32000	1000	30000	0	0	0	0	7	9	1	0
709.366614	23000	31000	41000	30000	0	0	0	0	8	9	10	0
710.370109	24000	31000	42000	40000	0	0	0	0	8	9	11	0
711.373319	23000	1000	42000	20000	0	0	0	0	8	1	11	0
712.376657	23000	30000	45000	0	0	0	0	0	8	8	11	0
713.379989	1000	31000	21000	0	0	0	0	0	1	9	11	0
714.383293	25000	31000	0	0	0	0	0	0	7	9	0	0
715.386654	29000	10000	0	0	0	0	0	0	9	9	0	0
716.389983	17000	0	0	0	0	0	0	0	9	0	0	0
717.393318	0	0	0	0	0	0	0	0	0	0	0	0
718.396649	0	0	0	0	0	0	0	0	0	0	0	0
719.399990	0	0	0	0	0	0	0	0	0	0	0	0
720.403310	0	0	0	0	0	0	0	0	0	0	0	0
721.406651	0	0	0	0	0	0	0	0	0	0	0	0
722.409983	0	0	0	0	0	0	0	0	0	0	0	0
723.413315	0	0	0	0	0	0	0	0	0	0	0	0
724.416652	0	0	0	0	0	0	0	0	0	0	0	0
725.419962	0	0	0	0	0	0	0	0	0	0	0	0
726.423319	0	0	0	0	0	0	0	0	0	0	0	0
727.426650	0	0	0	0	0	0	0	0	0	0	0	0
728.429983	0	0	0	0	0	0	0	0	0	0	0	0
729.433318	0	0	0	0	0	0	0	0	0	0	0	0
730.436651	0	0	0	0	0	0	0	0	0	0	0	0
731.439954	0	0	0	0	0	0	0	0	0	0	0	0
732.443313	0	0	0	0	0	0	0	0	0	0	0	0
733.446650	0	0	0	0	0	0	0	0	0	0	0	0
734.449988	0	0	0	0	0	0	0	0	0	0	0	0
735.453316	0	0	0	0	0	0	0	0	0	0	0	0
736.456653	0	0	0	0	0	0	0	0	0	0	0	0
737.459985	0	0	0	0	0	0	0	0	0	0	0	0
738.463316	0	0	0	0	0	0	0	0	0	0	0	0
739.466649	0	0	0	0	0	0	0	0	0	0	0	0
740.469986	0	0	0	0	0	0	0	0	0	0	0	0
741.473316	0	0	0	0	0	0	0	0	0	0	0	0
742.476610	0	0	0	0	0	0	0	0	0	0	0	0

A.2 Bandwidth Limited Client Sends Data to Server

Table A.3: Client Sending Data - Client Outgoing Bandwidth Limited

	Bandwidth (bytes per second)				Bytes Dropped				Retransmitted Packets			
Time Stamp	Group 0	Group 1	Group 2	Group 3	Group 0	Group 1	Group 2	Group 3	Group 0	Group 1	Group 2	Group 3
349.966688	0	0	0	0	0	0	0	0	0	0	0	0
350.970025	0	0	0	0	0	0	0	0	0	0	0	0
351.973358	0	0	0	0	0	0	0	0	0	0	0	0
352.976687	0	0	0	0	0	0	0	0	0	0	0	0
353.980029	0	0	0	0	0	0	0	0	0	0	0	0
354.983373	0	0	0	0	0	0	0	0	0	0	0	0
355.986726	0	0	0	0	0	0	0	0	0	0	0	0
356.990038	0	0	0	0	0	0	0	0	0	0	0	0
357.993371	0	0	0	0	0	0	0	0	0	0	0	0
358.996704	0	0	0	0	0	0	0	0	0	0	0	0
360.000033	0	0	0	0	0	0	0	0	0	0	0	0
361.003321	0	0	0	0	0	0	0	0	0	0	0	0
362.006670	0	0	0	0	0	0	0	0	0	0	0	0
363.010012	0	0	0	0	0	0	0	0	0	0	0	0
364.013338	0	0	0	0	0	0	0	0	0	0	0	0
365.016685	0	0	0	0	0	0	0	0	0	0	0	0
366.020021	0	0	0	0	0	0	0	0	0	0	0	0
367.023348	0	0	0	0	0	0	0	0	0	0	0	0
368.026663	0	0	0	0	0	0	0	0	0	0	0	0
369.030013	0	0	0	0	0	0	0	0	0	0	0	0
370.033321	0	0	0	0	0	0	0	0	0	0	0	0
371.036685	0	0	0	0	0	0	0	0	0	0	0	0
372.040018	0	0	0	0	0	0	0	0	0	0	0	0
373.043349	0	0	0	0	0	0	0	0	0	0	0	0
374.046684	0	0	0	0	0	0	0	0	0	0	0	0
375.049990	0	0	0	0	0	0	0	0	0	0	0	0
401.136662	10000	10000	10000	10000	0	0	0	0	0	0	0	0
402.139973	9000	9000	9000	9000	0	0	0	0	0	0	0	0
403.143352	10000	10000	10000	10000	0	0	0	0	0	0	0	0
404.146653	10000	10000	10000	10000	0	0	0	0	0	0	0	0
405.150002	10000	10000	10000	10000	0	0	0	0	0	0	0	0
406.153318	10000	10000	10000	10000	0	0	0	0	0	0	0	0
407.156654	10000	10000	10000	10000	0	0	0	0	0	0	0	0
408.159984	10000	10000	10000	10000	0	0	0	0	0	0	0	0
409.163353	10000	10000	10000	10000	0	0	0	0	0	0	0	0
410.166686	10000	10000	10000	10000	0	0	0	0	0	0	0	0
411.170019	10000	10000	10000	10000	0	0	0	0	0	0	0	0

412.173320	10000	10000	10000	10000	0	0	0	0	0	0	0	0
413.176666	10000	10000	10000	10000	0	0	0	0	0	0	0	0
414.179989	10000	10000	10000	10000	0	0	0	0	0	0	0	0
415.183321	10000	10000	10000	10000	0	0	0	0	0	0	0	0
416.186656	10000	10000	10000	10000	0	0	0	0	0	0	0	0
417.189983	10000	10000	10000	10000	0	0	0	0	0	0	0	0
418.193357	10000	10000	10000	10000	0	0	0	0	0	0	0	0
419.196692	10000	10000	10000	10000	0	0	0	0	0	0	0	0
420.200026	10000	10000	10000	10000	0	0	0	0	0	0	0	0
421.203349	10000	10000	10000	10000	0	0	0	0	0	0	0	0
422.206683	10000	10000	10000	10000	0	0	0	0	0	0	0	0
423.210023	10000	10000	10000	10000	0	0	0	0	0	0	0	0
424.213357	10000	10000	10000	10000	0	0	0	0	0	0	0	0
425.216644	10000	10000	11000	10000	0	0	0	0	0	0	0	0
426.219996	10000	10000	10000	10000	0	0	0	0	0	0	0	0
427.223315	11000	10000	10000	11000	0	0	0	0	0	0	0	0
428.226682	10000	10000	10000	10000	0	0	0	0	0	0	0	0
429.229982	10000	10000	10000	10000	0	0	0	0	0	0	0	0
430.233319	10000	10000	10000	10000	0	0	0	0	0	0	0	0
431.236654	15000	10000	10000	10000	0	0	0	0	0	0	0	0
432.240016	15000	10000	10000	10000	0	0	0	0	0	0	0	0
433.243352	15000	10000	10000	10000	0	0	0	0	0	0	0	0
434.246682	15000	10000	10000	10000	0	0	0	0	0	0	0	0
435.250009	15000	10000	10000	10000	0	0	0	0	0	0	0	0
436.253355	15000	10000	10000	10000	0	0	0	0	0	0	0	0
437.256651	15000	10000	10000	10000	0	0	0	0	0	0	0	0
438.260001	15000	10000	10000	10000	0	0	0	0	0	0	0	0
439.263335	15000	10000	10000	10000	0	0	0	0	0	0	0	0
440.266672	15000	10000	10000	10000	0	0	0	0	0	0	0	0
441.270006	15000	10000	10000	10000	0	0	0	0	0	0	0	0
442.273332	15000	10000	10000	10000	0	0	0	0	0	0	0	0
443.276672	15000	10000	10000	10000	0	0	0	0	0	0	0	0
444.279998	15000	10000	10000	10000	0	0	0	0	0	0	0	0
445.283325	15000	11000	10000	10000	0	0	0	0	0	0	0	0
446.286683	15000	9000	10000	10000	0	0	0	0	0	0	0	0
447.290012	15000	10000	10000	10000	0	0	0	0	0	0	0	0
448.293346	15000	10000	10000	10000	0	0	0	0	0	0	0	0
449.296665	15000	10000	10000	10000	0	0	0	0	0	0	0	0
450.300011	15000	10000	10000	10000	0	0	0	0	0	0	0	0
451.303345	15000	10000	10000	10000	0	0	0	0	0	0	0	0
452.306676	15000	10000	10000	10000	0	0	0	0	0	0	0	0
453.310013	15000	10000	10000	10000	0	0	0	0	0	0	0	0
454.313344	15000	10000	10000	10000	0	0	0	0	0	0	0	0
455.316678	15000	10000	10000	10000	0	0	0	0	0	0	0	0
456.319974	15000	11000	10000	10000	0	0	0	0	0	0	0	0
457.323327	15000	10000	10000	10000	0	0	0	0	0	0	0	0
458.326666	15000	10000	10000	10000	0	0	0	0	0	0	0	0
459.329997	15000	10000	10000	10000	0	0	0	0	0	0	0	0

460.333333	15000	10000	10000	10000	0	0	0	0	0	0	0	0
461.336661	20000	10000	10000	10000	0	0	0	0	0	0	0	0
462.340016	20000	10000	10000	10000	0	0	0	0	0	0	0	0
463.343327	20000	10000	10000	10000	0	0	0	0	0	0	0	0
464.346683	20000	10000	10000	10000	0	0	0	0	0	0	0	0
465.349988	20000	10000	10000	10000	0	0	0	0	0	0	0	0
466.353326	20000	10000	10000	10000	0	0	0	0	0	0	0	0
467.356668	20000	10000	10000	10000	0	0	0	0	0	0	0	0
468.359985	20000	10000	10000	10000	0	0	0	0	0	0	0	0
469.363326	20000	10000	10000	10000	0	0	0	0	0	0	0	0
470.366684	20000	10000	10000	10000	0	0	0	0	0	0	0	0
471.370015	20000	10000	10000	10000	0	0	0	0	0	0	0	0
472.373338	20000	10000	10000	10000	0	0	0	0	0	0	0	0
473.376672	20000	10000	10000	10000	0	0	0	0	0	0	0	0
474.380013	20000	10000	10000	10000	0	0	0	0	0	0	0	0
475.383361	20000	10000	10000	10000	0	0	0	0	0	0	0	0
476.386656	20000	10000	10000	10000	0	0	0	0	0	0	0	0
477.390026	20000	10000	10000	10000	0	0	0	0	0	0	0	0
478.393370	20000	10000	10000	10000	0	0	0	0	0	0	0	0
479.396669	20000	10000	10000	10000	0	0	0	0	0	0	0	0
480.400038	20000	10000	10000	10000	0	0	0	0	0	0	0	0
481.403367	20000	10000	10000	10000	0	0	0	0	0	0	0	0
482.406658	20000	10000	10000	10000	0	0	0	0	0	0	0	0
483.409988	20000	10000	10000	10000	0	0	0	0	0	0	0	0
484.413373	20000	10000	10000	10000	0	0	0	0	0	0	0	0
485.416711	20000	10000	10000	10000	0	0	0	0	0	0	0	0
486.420021	20000	10000	10000	10000	0	0	0	0	0	0	0	0
487.423359	20000	10000	10000	10000	0	0	0	0	0	0	0	0
488.426686	20000	10000	10000	10000	0	0	0	0	0	0	0	0
489.429998	20000	10000	10000	10000	0	0	0	0	0	0	0	0
490.433333	20000	10000	10000	10000	0	0	0	0	0	0	0	0
491.436705	25000	10000	10000	10000	0	0	0	0	0	0	0	0
492.440026	25000	10000	10000	10000	0	0	0	0	0	0	0	0
493.443334	25000	10000	10000	10000	0	0	0	0	0	0	0	0
494.446690	25000	10000	10000	10000	0	0	0	0	0	0	0	0
495.450023	25000	10000	10000	10000	0	0	0	0	0	0	0	0
496.453352	25000	10000	10000	10000	0	0	0	0	0	0	0	0
497.456671	25000	10000	10000	10000	0	0	0	0	0	0	0	0
498.459983	25000	10000	10000	10000	0	0	0	0	0	0	0	0
499.463369	25000	10000	10000	10000	0	0	0	0	0	0	0	0
500.466697	25000	10000	10000	10000	0	0	0	0	0	0	0	0
501.470035	25000	10000	10000	10000	0	0	0	0	0	0	0	0
502.473328	25000	10000	10000	10000	0	0	0	0	0	0	0	0
503.476681	25000	10000	10000	10000	0	0	0	0	0	0	0	0
504.479992	25000	10000	10000	10000	0	0	0	0	0	0	0	0
505.483322	25000	10000	10000	10000	0	0	0	0	0	0	0	0
506.486664	25000	10000	10000	10000	0	0	0	0	0	0	0	0
507.490035	25000	10000	10000	10000	0	0	0	0	0	0	0	0

508.493333	25000	10000	10000	10000	0	0	0	0	0	0	0	0
509.496670	25000	10000	10000	10000	0	0	0	0	0	0	0	0
510.500002	25000	10000	10000	10000	0	0	0	0	0	0	0	0
511.503337	25000	10000	10000	10000	0	0	0	0	0	0	0	0
512.506671	25000	10000	10000	10000	0	0	0	0	0	0	0	0
513.510002	25000	10000	10000	10000	0	0	0	0	0	0	0	0
514.513338	25000	10000	10000	10000	0	0	0	0	0	0	0	0
515.516643	25000	10000	10000	10000	0	0	0	0	0	0	0	0
516.520003	25000	10000	10000	10000	0	0	0	0	0	0	0	0
517.523337	25000	10000	10000	10000	0	0	0	0	0	0	0	0
518.526657	25000	10000	10000	10000	0	0	0	0	0	0	0	0
519.529991	25000	10000	10000	10000	0	0	0	0	0	0	0	0
520.533326	25000	10000	10000	10000	0	0	0	0	0	0	0	0
521.536659	30000	10000	10000	10000	0	0	0	0	0	0	0	0
522.540006	30000	10000	10000	10000	0	0	0	0	0	0	0	0
523.543310	30000	10000	10000	10000	0	0	0	0	0	0	0	0
524.546655	30000	10000	10000	10000	0	0	0	0	0	0	0	0
525.549988	30000	10000	10000	10000	0	0	0	0	0	0	0	0
526.553325	30000	10000	10000	10000	0	0	0	0	0	0	0	0
527.556659	30000	10000	10000	10000	0	0	0	0	0	0	0	0
528.559989	30000	10000	10000	10000	0	0	0	0	0	0	0	0
529.563338	30000	10000	10000	10000	0	0	0	0	0	0	0	0
530.566671	30000	10000	10000	10000	0	0	0	0	0	0	0	0
531.570009	30000	10000	10000	10000	0	0	0	0	0	0	0	0
532.573300	30000	10000	10000	10000	0	0	0	0	0	0	0	0
533.576663	30000	10000	10000	10000	0	0	0	0	0	0	0	0
534.579991	30000	10000	10000	10000	0	0	0	0	0	0	0	0
535.583339	30000	10000	10000	10000	0	0	0	0	0	0	0	0
536.586644	30000	10000	10000	10000	0	0	0	0	0	0	0	0
537.590017	30000	10000	10000	10000	0	0	0	0	0	0	0	0
538.593347	30000	10000	10000	10000	0	0	0	0	0	0	0	0
539.596675	30000	10000	10000	10000	0	0	0	0	0	0	0	0
540.600007	30000	10000	10000	10000	0	0	0	0	0	0	0	0
541.603346	30000	10000	10000	10000	0	0	0	0	0	0	0	0
542.606680	30000	10000	10000	10000	0	0	0	0	0	0	0	0
543.610007	30000	10000	10000	10000	0	0	0	0	0	0	0	0
544.613343	30000	10000	10000	10000	0	0	0	0	0	0	0	0
545.616673	30000	10000	10000	10000	0	0	0	0	0	0	0	0
546.620012	30000	10000	10000	10000	0	0	0	0	0	0	0	0
547.623347	30000	10000	10000	10000	0	0	0	0	0	0	0	0
548.626640	30000	10000	10000	10000	0	0	0	0	0	0	0	0
549.629962	30000	10000	10000	10000	0	0	0	0	0	0	0	0
550.633355	30000	10000	10000	10000	0	0	0	0	0	0	0	0
551.636674	30000	15000	10000	10000	0	0	0	0	0	0	0	0
552.640011	30000	15000	10000	10000	0	0	0	0	0	0	0	0
553.643345	30000	15000	10000	10000	0	0	0	0	0	0	0	0
554.646670	30000	15000	10000	10000	0	0	0	0	0	0	0	0
555.650010	30000	15000	10000	10000	0	0	0	0	0	0	0	0

556.653335	30000	15000	10000	10000	0	0	0	0	0	0	0	0
557.656679	30000	15000	10000	10000	0	0	0	0	0	0	0	0
558.659992	30000	15000	10000	10000	0	0	0	0	0	0	0	0
559.663336	30000	15000	10000	10000	0	0	0	0	0	0	0	0
560.666659	30000	15000	10000	10000	0	0	0	0	0	0	0	0
561.670018	30000	15000	10000	10000	0	0	0	0	0	0	0	0
562.673345	30000	15000	10000	10000	0	0	0	0	0	0	0	0
563.676672	30000	15000	10000	10000	0	0	0	0	0	0	0	0
564.679999	30000	15000	10000	10000	0	0	0	0	0	0	0	0
565.683347	30000	15000	10000	10000	0	0	0	0	0	0	0	0
566.686631	30000	15000	10000	10000	0	0	0	0	0	0	0	0
567.690001	30000	15000	10000	10000	0	0	0	0	0	0	0	0
568.693320	30000	15000	10000	10000	0	0	0	0	0	0	0	0
569.696663	30000	15000	10000	10000	0	0	0	0	0	0	0	0
570.699999	30000	15000	10000	10000	0	0	0	0	0	0	0	0
571.703330	30000	15000	10000	10000	0	0	0	0	0	0	0	0
572.706666	30000	15000	10000	10000	0	0	0	0	0	0	0	0
573.709997	30000	15000	10000	10000	0	0	0	0	0	0	0	0
574.713331	30000	15000	10000	10000	0	0	0	0	0	0	0	0
575.716670	30000	15000	10000	10000	0	0	0	0	0	0	0	0
576.720002	30000	15000	10000	10000	0	0	0	0	0	0	0	0
577.723333	30000	15000	10000	10000	0	0	0	0	0	0	0	0
578.726669	30000	15000	10000	10000	0	0	0	0	0	0	0	0
579.730001	30000	15000	10000	10000	0	0	0	0	0	0	0	0
580.733333	30000	15000	10000	10000	0	0	0	0	0	0	0	0
581.736665	30000	20000	10000	10000	0	0	0	0	0	0	0	0
582.739996	30000	20000	10000	10000	0	0	0	0	0	0	0	0
583.743289	30000	20000	10000	10000	0	0	0	0	0	0	0	0
584.746650	30000	20000	10000	10000	0	0	0	0	0	0	0	0
585.750003	30000	20000	10000	10000	0	0	0	0	0	0	0	0
586.753315	30000	20000	10000	10000	0	0	0	0	0	0	0	0
587.756674	30000	20000	10000	10000	0	0	0	0	0	0	0	0
588.759975	30000	20000	10000	10000	0	0	0	0	0	0	0	0
589.763331	30000	20000	10000	10000	0	0	0	0	0	0	0	0
590.766649	30000	20000	10000	10000	0	0	0	0	0	0	0	0
591.769998	30000	20000	10000	10000	0	0	0	0	0	0	0	0
592.773313	30000	20000	10000	10000	0	0	0	0	0	0	0	0
593.776674	30000	20000	10000	10000	0	0	0	0	0	0	0	0
594.779992	30000	20000	10000	10000	0	0	0	0	0	0	0	0
595.783339	30000	20000	10000	10000	0	0	0	0	0	0	0	0
596.786658	30000	20000	10000	10000	0	0	0	0	0	0	0	0
597.790007	30000	20000	10000	10000	0	0	0	0	0	0	0	0
598.793347	30000	20000	10000	10000	0	0	0	0	0	0	0	0
599.796663	30000	20000	10000	10000	0	0	0	0	0	0	0	0
600.800443	30000	20000	10000	10000	0	0	0	0	0	0	0	0
601.803328	30000	20000	10000	10000	0	0	0	0	0	0	0	0
602.806663	30000	20000	10000	10000	0	0	0	0	0	0	0	0
603.809995	30000	20000	10000	10000	0	0	0	0	0	0	0	0

604.813331	30000	20000	10000	10000	0	0	0	0	0	0	0	0
605.816666	30000	20000	10000	10000	0	0	0	0	0	0	0	0
606.819995	30000	20000	10000	10000	0	0	0	0	0	0	0	0
607.823332	30000	20000	10000	10000	0	0	0	0	0	0	0	0
608.826635	30000	20000	10000	10000	0	0	0	0	0	0	0	0
609.829996	30000	20000	10000	10000	0	0	0	0	0	0	0	0
610.833333	30000	20000	10000	10000	0	0	0	0	0	0	0	0
611.836663	30000	25000	10000	10000	0	0	0	0	0	0	0	0
612.839997	30000	25000	10000	10000	0	0	0	0	0	0	0	0
613.843335	30000	25000	10000	10000	0	0	0	0	0	0	0	0
614.846664	30000	25000	10000	10000	0	0	0	0	0	0	0	0
615.849997	30000	25000	10000	10000	0	0	0	0	0	0	0	0
616.853334	30000	25000	10000	10000	0	0	0	0	0	0	0	0
617.856665	30000	25000	10000	10000	0	0	0	0	0	0	0	0
618.860000	30000	25000	10000	10000	0	0	0	0	0	0	0	0
619.863334	30000	25000	10000	10000	0	0	0	0	0	0	0	0
620.866665	30000	25000	10000	10000	0	0	0	0	0	0	0	0
621.870003	30000	25000	10000	10000	0	0	0	0	0	0	0	0
622.873336	30000	25000	10000	10000	0	0	0	0	0	0	0	0
623.876661	30000	25000	10000	10000	0	0	0	0	0	0	0	0
624.879998	30000	25000	10000	10000	0	0	0	0	0	0	0	0
625.883330	30000	25000	10000	10000	0	0	0	0	0	0	0	0
626.886677	30000	25000	10000	10000	0	0	0	0	0	0	0	0
627.889998	30000	25000	10000	10000	0	0	0	0	0	0	0	0
628.893319	30000	25000	10000	10000	0	0	0	0	0	0	0	0
629.896660	30000	25000	10000	10000	0	0	0	0	0	0	0	0
630.899990	30000	25000	10000	10000	0	0	0	0	0	0	0	0
631.903326	30000	25000	10000	10000	0	0	0	0	0	0	0	0
632.906661	30000	25000	10000	10000	0	0	0	0	0	0	0	0
633.909992	30000	25000	10000	10000	0	0	0	0	0	0	0	0
634.913327	30000	25000	10000	10000	0	0	0	0	0	0	0	0
635.916660	30000	25000	10000	10000	0	0	0	0	0	0	0	0
636.919995	30000	25000	10000	10000	0	0	0	0	0	0	0	0
637.923327	30000	25000	10000	10000	0	0	0	0	0	0	0	0
638.926662	30000	25000	10000	10000	0	0	0	0	0	0	0	0
639.929991	30000	25000	10000	10000	0	0	0	0	0	0	0	0
640.933327	30000	25000	10000	10000	0	0	0	0	0	0	0	0
641.936660	30000	30000	10000	10000	0	0	0	0	0	0	0	0
642.939994	30000	30000	10000	10000	0	0	0	0	0	0	0	0
643.943328	30000	30000	10000	10000	0	0	0	0	0	0	0	0
644.946635	30000	30000	10000	10000	0	0	0	0	0	0	0	0
645.949996	30000	30000	10000	10000	0	0	0	0	0	0	0	0
646.953331	30000	30000	10000	10000	0	0	0	0	0	0	0	0
647.956659	30000	30000	10000	10000	0	0	0	0	0	0	0	0
648.959991	30000	30000	10000	10000	0	0	0	0	0	0	0	0
649.963325	30000	30000	10000	10000	0	0	0	0	0	0	0	0
650.966635	30000	30000	10000	10000	0	0	0	0	0	0	0	0
651.969991	30000	30000	10000	10000	0	0	0	0	0	0	0	0

652.973338	30000	30000	10000	10000	0	0	0	0	0	0	0	0
653.976659	30000	30000	10000	10000	0	0	0	0	0	0	0	0
654.979981	30000	30000	10000	10000	0	0	0	0	0	0	0	0
655.983335	30000	30000	10000	10000	0	0	0	0	0	0	0	0
656.986677	30000	30000	10000	10000	0	0	0	0	0	0	0	0
657.989995	30000	30000	10000	10000	0	0	0	0	0	0	0	0
658.993331	30000	30000	10000	10000	0	0	0	0	0	0	0	0
659.996658	30000	30000	10000	10000	0	0	0	0	0	0	0	0
661.000008	30000	30000	10000	10000	0	0	0	0	0	0	0	0
662.003322	30000	30000	10000	10000	0	0	0	0	0	0	0	0
663.006669	30000	30000	10000	10000	0	0	0	0	0	0	0	0
664.009957	30000	30000	10000	10000	0	0	0	0	0	0	0	0
665.013327	30000	30000	10000	10000	0	0	0	0	0	0	0	0
666.016623	30000	30000	10000	10000	0	0	0	0	0	0	0	0
667.019991	30000	30000	10000	10000	0	0	0	0	0	0	0	0
668.023322	30000	30000	10000	10000	0	0	0	0	0	0	0	0
669.026610	30000	30000	10000	10000	0	0	0	0	0	0	0	0
670.030005	30000	30000	10000	10000	0	0	0	0	0	0	0	0
671.033334	30000	30000	10000	10000	0	0	0	0	0	0	0	0
672.036656	30000	30000	15000	10000	0	0	0	0	0	0	0	0
673.039995	30000	30000	15000	10000	0	0	0	0	0	0	0	0
674.043343	30000	30000	15000	10000	0	0	0	0	0	0	0	0
675.046673	30000	30000	15000	10000	0	0	0	0	0	0	0	0
676.050073	30000	30000	23000	10000	0	0	0	0	0	0	0	0
677.053678	30000	30000	21000	18000	0	0	0	0	0	0	0	0
678.056613	13000	20000	16000	11000	30000	9000	0	0	0	0	0	0
679.059944	13000	23000	15000	11000	11000	12000	0	0	24	9	0	0
680.063276	0	26000	15000	9000	0	11000	0	0	0	12	0	0
681.066612	15000	26000	15000	10000	8000	11000	0	0	17	11	0	0
682.069963	0	1000	15000	11000	0	0	0	0	0	1	0	0
683.073280	0	33000	15000	9000	0	11000	0	0	0	10	0	0
684.076613	0	38000	15000	11000	0	12000	0	0	0	11	0	0
685.079944	11000	42000	15000	9000	8000	12000	0	0	8	12	0	0
686.083278	11000	45000	15000	10000	8000	12000	0	0	8	12	0	0
687.086615	1000	46000	15000	10000	0	0	0	0	1	12	0	0
688.089944	14000	30000	15000	10000	8000	0	0	0	7	0	0	0
689.093323	17000	30000	15000	11000	9000	0	0	0	8	0	0	0
690.096615	20000	30000	15000	10000	9000	0	0	0	9	0	0	0
691.099985	23000	30000	15000	10000	9000	0	0	0	9	0	0	0
692.103301	26000	30000	15000	9000	11000	0	0	0	9	0	0	0
693.106624	1000	30000	15000	10000	0	0	0	0	1	0	0	0
694.109973	33000	30000	15000	10000	11000	0	0	0	10	0	0	0
695.113326	36000	30000	15000	11000	11000	0	0	0	11	0	0	0
696.116660	39000	30000	15000	10000	12000	0	0	0	11	0	0	0
697.119987	42000	30000	15000	9000	12000	0	0	0	12	0	0	0
698.123319	43000	30000	15000	11000	12000	0	0	0	12	0	0	0
699.126653	1000	23000	15000	9000	0	6000	0	0	1	0	0	0
700.129957	10000	27000	15000	11000	8000	7000	0	0	11	6	0	0

701.133309	0	23000	19000	10000	0	9000	0	0	0	7	0	0
702.136628	12000	23000	20000	9000	8000	9000	0	0	8	9	0	0
703.139959	15000	23000	20000	10000	8000	9000	0	0	8	9	0	0
704.143320	18000	1000	20000	10000	9000	0	0	0	8	1	0	0
705.146657	1000	26000	20000	10000	0	11000	0	0	1	8	0	0
706.149979	21000	26000	20000	10000	9000	11000	0	0	8	11	0	0
707.153557	23000	26000	20000	10000	9000	11000	0	0	9	11	0	0
708.156648	24000	26000	20000	10000	9000	11000	0	0	9	11	0	0
709.159944	26000	26000	20000	10000	11000	11000	0	0	9	11	0	0
710.163279	27000	26000	20000	10000	11000	11000	0	0	11	11	0	0
711.166646	1000	1000	20000	11000	0	0	0	0	1	1	0	0
712.169996	31000	25000	20000	10000	11000	11000	0	0	10	10	0	0
713.173314	33000	25000	20000	9000	11000	9000	0	0	11	11	0	0
714.176752	34000	25000	20000	11000	11000	9000	0	0	11	9	0	0
715.179992	36000	25000	20000	10000	11000	9000	0	0	11	9	0	0
716.183281	1000	26000	20000	10000	0	11000	0	0	1	9	0	0
717.186637	38000	1000	20000	10000	12000	0	0	0	10	1	0	0
718.189986	41000	24000	20000	10000	12000	9000	0	0	12	10	0	0
719.193319	43000	24000	21000	10000	12000	9000	0	0	12	9	0	0
720.196648	44000	24000	19000	10000	12000	9000	0	0	12	9	0	0
721.199991	46000	24000	20000	9000	12000	9000	0	0	12	9	0	0
722.203303	13000	20000	20000	11000	8000	9000	0	0	12	9	0	0
723.206638	1000	1000	20000	10000	0	0	0	0	1	1	0	0
724.209979	20000	20000	21000	10000	9000	9000	0	0	7	8	0	0
725.213278	13000	20000	19000	10000	8000	9000	0	0	9	9	0	0
726.216621	16000	20000	21000	10000	8000	9000	0	0	8	9	0	0
727.219960	19000	20000	19000	10000	9000	9000	0	0	8	9	0	0
728.223312	22000	20000	21000	10000	9000	9000	0	0	9	9	0	0
729.226649	1000	1000	19000	10000	0	0	0	0	1	1	0	0
730.229980	29000	20000	20000	10000	11000	9000	0	0	8	8	0	0
731.233326	13000	20000	26000	9000	8000	9000	0	0	11	9	0	0
732.236659	16000	20000	25000	11000	8000	9000	0	0	8	9	0	0
733.239984	19000	20000	25000	10000	9000	9000	0	0	8	9	0	0
734.243319	1000	1000	25000	9000	0	0	0	0	1	1	0	0
735.246634	23000	20000	25000	11000	9000	9000	0	0	8	8	0	0
736.249996	13000	20000	25000	9000	8000	9000	0	0	9	9	0	0
737.253322	16000	20000	24000	11000	8000	9000	0	0	8	9	0	0
738.256657	19000	20000	26000	9000	9000	9000	0	0	8	9	0	0
739.259983	13000	20000	25000	11000	8000	9000	0	0	9	9	0	0
740.263313	1000	1000	25000	9000	0	0	0	0	1	1	0	0
741.266643	20000	20000	25000	11000	9000	9000	0	0	7	8	0	0
742.269945	13000	20000	25000	10000	8000	9000	0	0	9	9	0	0
743.273308	16000	20000	25000	10000	8000	9000	0	0	8	9	0	0
744.276609	19000	20000	25000	10000	9000	9000	0	0	8	9	0	0
745.279984	20000	20000	25000	10000	9000	9000	0	0	9	9	0	0
746.283302	1000	1000	25000	10000	0	0	0	0	1	1	0	0
747.286616	24000	20000	25000	10000	9000	9000	0	0	8	8	0	0
748.289987	13000	20000	25000	10000	8000	9000	0	0	9	9	0	0

749.293273	16000	20000	25000	10000	8000	9000	0	0	8	9	0	0
750.296643	19000	20000	25000	10000	9000	9000	0	0	8	9	0	0
751.299977	1000	1000	25000	10000	0	0	0	0	1	1	0	0
752.303283	23000	20000	25000	10000	9000	9000	0	0	8	8	0	0
753.306617	13000	20000	25000	10000	8000	9000	0	0	9	9	0	0
754.309981	16000	20000	25000	10000	8000	9000	0	0	8	9	0	0
755.313315	19000	20000	25000	10000	9000	9000	0	0	8	9	0	0
756.316607	20000	1000	25000	10000	9000	0	0	0	9	1	0	0
757.319966	1000	21000	25000	10000	0	9000	0	0	1	8	0	0
758.323309	21000	21000	25000	10000	9000	9000	0	0	8	9	0	0
759.326651	23000	21000	25000	10000	9000	9000	0	0	9	9	0	0
760.329980	24000	21000	25000	10000	9000	9000	0	0	9	9	0	0
761.333299	10000	20000	30000	10000	8000	8000	0	0	9	9	0	0
762.336650	1000	21000	30000	10000	0	9000	0	0	1	8	0	0
763.339990	11000	21000	30000	10000	8000	9000	0	0	7	9	0	0
764.343317	13000	21000	30000	10000	8000	9000	0	0	8	9	0	0
765.346648	14000	21000	30000	10000	8000	9000	0	0	8	9	0	0
766.349980	16000	1000	30000	10000	8000	0	0	0	8	1	0	0
767.353314	1000	22000	30000	10000	0	9000	0	0	1	8	0	0
768.356652	18000	22000	30000	10000	9000	9000	0	0	7	9	0	0
769.359937	19000	22000	30000	10000	9000	9000	0	0	9	9	0	0
770.363320	21000	22000	30000	10000	9000	9000	0	0	9	9	0	0
771.366652	22000	22000	30000	10000	9000	9000	0	0	9	9	0	0
772.369987	24000	1000	30000	10000	9000	0	0	0	9	1	0	0
773.373313	1000	24000	30000	10000	0	9000	0	0	1	8	0	0
774.376645	25000	24000	30000	10000	11000	9000	0	0	8	9	0	0
775.379970	26000	24000	30000	10000	11000	9000	0	0	11	9	0	0
776.383329	28000	24000	30000	10000	11000	9000	0	0	11	9	0	0
777.386626	29000	24000	30000	10000	11000	9000	0	0	11	9	0	0
778.389939	31000	24000	30000	10000	11000	9000	0	0	11	9	0	0
779.393303	1000	1000	30000	10000	0	0	0	0	1	1	0	0
780.396677	36000	22000	30000	10000	11000	9000	0	0	10	8	0	0
781.399975	37000	22000	30000	10000	12000	9000	0	0	11	9	0	0
782.403342	37000	22000	30000	10000	11000	9000	0	0	12	9	0	0
783.406656	37000	22000	30000	10000	11000	8000	0	0	11	9	0	0
784.409938	37000	22000	30000	10000	11000	7000	0	0	11	8	0	0
785.413333	11000	0	30000	10000	7000	0	0	0	11	0	0	0
786.416626	14000	20000	30000	10000	8000	8000	0	0	7	7	0	0
787.419982	11000	20000	30000	10000	8000	8000	0	0	8	8	0	0
788.423346	13000	20000	30000	10000	8000	8000	0	0	8	8	0	0
789.426603	14000	20000	30000	10000	7000	8000	0	0	8	8	0	0
790.429967	16000	1000	30000	10000	8000	0	0	0	7	1	0	0
791.433341	16000	21000	30000	15000	8000	8000	0	0	8	7	0	0
792.436662	16000	22000	30000	15000	8000	7000	0	0	8	8	0	0
793.439981	16000	24000	30000	15000	8000	8000	0	0	8	7	0	0
794.443295	1000	25000	30000	15000	0	8000	0	0	1	8	0	0
795.446646	17000	1000	30000	15000	9000	0	0	0	7	1	0	0
796.449959	17000	27000	30000	15000	9000	8000	0	0	9	7	0	0

797.453339	17000	29000	30000	15000	9000	8000	0	0	9	8	0	0
798.456643	17000	30000	30000	15000	9000	7000	0	0	9	8	0	0
799.459974	17000	32000	30000	15000	9000	8000	0	0	9	7	0	0
800.463279	1000	33000	30000	15000	0	8000	0	0	1	8	0	0
801.466645	19000	1000	30000	15000	9000	0	0	0	8	1	0	0
802.469987	19000	34000	30000	15000	9000	8000	0	0	9	7	0	0
803.473270	19000	34000	30000	15000	9000	7000	0	0	9	8	0	0
804.476642	19000	34000	30000	15000	8000	7000	0	0	9	7	0	0
805.479978	19000	34000	30000	15000	8000	7000	0	0	8	7	0	0
806.483299	1000	34000	30000	15000	0	8000	0	0	1	7	0	0
807.486630	19000	1000	30000	15000	8000	0	0	0	7	1	0	0
808.489978	19000	34000	30000	15000	8000	7000	0	0	8	7	0	0
809.493273	19000	34000	30000	15000	8000	7000	0	0	8	7	0	0
810.496609	19000	34000	30000	15000	8000	7000	0	0	8	7	0	0
811.499974	19000	34000	30000	15000	8000	7000	0	0	8	7	0	0
812.503306	1000	34000	30000	15000	0	8000	0	0	1	7	0	0
813.506640	20000	1000	30000	15000	8000	0	0	0	7	1	0	0
814.509932	20000	33000	30000	15000	8000	7000	0	0	8	7	0	0
815.513316	20000	33000	30000	15000	8000	8000	0	0	8	7	0	0
816.516647	20000	33000	30000	15000	8000	8000	0	0	8	8	0	0
817.519973	20000	33000	30000	15000	7000	8000	0	0	8	8	0	0
818.523311	20000	33000	30000	15000	8000	8000	0	0	7	8	0	0
819.526643	1000	1000	30000	15000	0	0	0	0	1	1	0	0
820.529969	18000	35000	30000	15000	8000	8000	0	0	7	7	0	0
821.533305	18000	35000	29000	17000	7000	8000	1000	3000	8	8	0	0
822.536647	10000	21000	30000	23000	7000	8000	1000	0	7	8	1	3
823.539979	1000	22000	31000	20000	0	8000	0	0	1	8	1	0
824.543306	11000	22000	30000	20000	7000	8000	0	0	6	8	0	0
825.546610	12000	1000	30000	20000	7000	0	0	0	7	1	0	0
826.549975	12000	23000	30000	20000	7000	8000	0	0	7	7	0	0
827.553289	0	25000	30000	20000	0	8000	0	0	0	8	0	0
828.556657	14000	25000	30000	20000	8000	8000	0	0	7	8	0	0
829.559952	15000	25000	30000	20000	8000	8000	0	0	8	8	0	0
830.563306	17000	25000	30000	20000	8000	8000	0	0	8	8	0	0
831.566602	18000	25000	30000	20000	8000	8000	0	0	8	8	0	0
832.569968	1000	1000	30000	20000	0	0	0	0	1	1	0	0
833.573300	22000	24000	30000	20000	8000	8000	0	0	7	7	0	0
834.576648	24000	24000	30000	20000	8000	8000	0	0	8	8	0	0
835.579936	24000	24000	30000	20000	8000	7000	0	0	8	8	0	0
836.583315	24000	24000	30000	20000	8000	8000	0	0	8	7	0	0
837.586599	24000	24000	30000	20000	7000	8000	0	0	8	8	0	0
838.589957	1000	1000	30000	20000	0	0	0	0	1	1	0	0
839.593305	26000	22000	30000	20000	8000	8000	0	0	6	7	0	0
840.596627	26000	22000	30000	20000	8000	8000	0	0	8	8	0	0
841.599967	26000	22000	30000	20000	8000	8000	0	0	8	8	0	0
842.603320	26000	22000	30000	20000	8000	8000	0	0	8	8	0	0
843.606609	26000	22000	30000	20000	8000	8000	0	0	8	8	0	0
844.609974	1000	1000	30000	20000	0	0	0	0	1	1	0	0

845.613289	27000	21000	30000	20000	8000	8000	0	0	7	7	0	0
846.616638	27000	21000	30000	20000	8000	8000	0	0	8	8	0	0
847.619988	27000	21000	30000	20000	8000	8000	0	0	8	8	0	0
848.623306	27000	21000	30000	20000	8000	8000	0	0	8	8	0	0
849.626600	27000	21000	30000	20000	8000	8000	0	0	8	8	0	0
850.629965	27000	1000	30000	20000	8000	0	0	0	8	1	0	0
851.633324	1000	20000	30000	25000	0	8000	0	0	1	7	0	0
852.636630	11000	20000	30000	25000	7000	8000	0	0	7	8	0	0
853.639934	12000	20000	30000	25000	7000	8000	0	0	7	8	0	0
854.643317	12000	22000	30000	25000	7000	8000	0	0	7	8	0	0
855.646649	0	23000	30000	25000	0	8000	0	0	0	8	0	0
856.649966	14000	1000	30000	25000	8000	0	0	0	7	1	0	0
857.653267	14000	24000	30000	25000	8000	8000	0	0	8	7	0	0
858.656617	14000	26000	30000	25000	8000	8000	0	0	8	8	0	0
859.659976	14000	27000	30000	25000	8000	8000	0	0	8	8	0	0
860.663331	1000	29000	30000	25000	0	8000	0	0	1	8	0	0
861.666633	15000	29000	30000	25000	8000	8000	0	0	7	8	0	0
862.669974	11000	1000	30000	25000	8000	0	0	0	8	1	0	0
863.673323	11000	21000	30000	25000	8000	8000	0	0	8	7	0	0
864.676621	11000	23000	30000	25000	8000	8000	0	0	8	8	0	0
865.679984	11000	24000	30000	25000	8000	8000	0	0	8	8	0	0
866.683298	1000	26000	30000	25000	0	8000	0	0	1	8	0	0
867.686617	12000	26000	30000	25000	8000	7000	0	0	7	8	0	0
868.689986	13000	1000	30000	25000	8000	0	0	0	8	1	0	0
869.693295	13000	28000	30000	25000	8000	8000	0	0	8	6	0	0
870.696635	13000	30000	30000	25000	8000	8000	0	0	8	8	0	0
871.699939	13000	31000	30000	25000	8000	8000	0	0	8	8	0	0
872.703264	10000	21000	30000	25000	7000	8000	0	0	8	8	0	0
873.706607	0	23000	30000	25000	0	8000	0	0	0	8	0	0
874.709967	11000	1000	30000	25000	8000	0	0	0	7	1	0	0
875.713300	11000	24000	30000	25000	8000	8000	0	0	8	7	0	0
876.716636	11000	25000	30000	25000	8000	8000	0	0	8	8	0	0
877.719929	11000	27000	30000	25000	8000	8000	0	0	8	8	0	0
878.723291	1000	28000	30000	25000	0	7000	0	0	1	8	0	0
879.726636	13000	28000	30000	25000	8000	8000	0	0	7	7	0	0
880.729974	15000	1000	30000	25000	8000	0	0	0	8	1	0	0
881.733312	9000	30000	30000	30000	8000	8000	0	0	8	7	0	0
882.736645	10000	21000	30000	30000	7000	8000	0	0	8	8	0	0
883.739965	1000	23000	30000	30000	0	8000	0	0	1	8	0	0
884.743300	11000	23000	30000	30000	7000	8000	0	0	6	8	0	0
885.746636	12000	23000	30000	30000	7000	8000	0	0	7	8	0	0
886.749928	14000	1000	30000	30000	7000	0	0	0	7	1	0	0
887.753266	0	24000	30000	30000	0	8000	0	0	0	7	0	0
888.756645	13000	25000	30000	30000	8000	8000	0	0	7	8	0	0
889.759959	13000	25000	30000	30000	8000	8000	0	0	8	8	0	0
890.763309	13000	25000	30000	30000	8000	8000	0	0	8	8	0	0
891.766612	13000	25000	30000	30000	8000	8000	0	0	8	8	0	0
892.769983	1000	1000	30000	30000	0	0	0	0	1	1	0	0

893.773296	11000	27000	30000	30000	7000	10000	0	0	7	9	0	0
894.776644	11000	0	30000	30000	8000	0	0	0	7	0	0	0
895.780307	13000	1000	30000	30000	8000	0	0	0	8	1	0	0
896.783288	13000	25000	30000	30000	8000	8000	0	0	8	7	0	0
897.786657	1000	25000	30000	30000	0	8000	0	0	1	8	0	0
898.789973	14000	25000	30000	30000	8000	8000	0	0	7	8	0	0
899.793321	11000	20000	30000	30000	8000	8000	0	0	8	8	0	0
900.796594	13000	20000	30000	30000	8000	8000	0	0	8	8	0	0
901.799969	14000	1000	30000	30000	8000	0	0	0	8	1	0	0
902.803302	14000	21000	30000	30000	8000	8000	0	0	8	7	0	0
903.806635	1000	23000	30000	30000	0	8000	0	0	1	8	0	0
904.809963	15000	23000	30000	30000	8000	8000	0	0	7	8	0	0
905.813297	15000	23000	30000	30000	8000	8000	0	0	8	8	0	0
906.816635	15000	23000	30000	30000	8000	8000	0	0	8	8	0	0
907.819967	15000	1000	30000	30000	8000	0	0	0	8	1	0	0
908.823286	15000	23000	30000	30000	8000	7000	0	0	8	7	0	0
909.827710	1000	23000	30000	30000	0	8000	0	0	1	7	0	0
910.829966	16000	23000	30000	30000	8000	8000	0	0	7	8	0	0
911.833298	11000	20000	30000	30000	7000	8000	0	0	8	8	0	0
912.836637	13000	20000	30000	30000	8000	8000	0	0	7	8	0	0
913.839969	14000	1000	30000	30000	8000	0	0	0	8	1	0	0
914.843310	14000	21000	30000	30000	7000	8000	0	0	8	7	0	0
915.846647	0	23000	30000	30000	0	8000	0	0	0	8	0	0
916.849966	15000	23000	30000	30000	8000	8000	0	0	7	8	0	0
917.853309	15000	23000	30000	30000	8000	8000	0	0	8	8	0	0
918.856604	15000	23000	30000	30000	8000	8000	0	0	8	8	0	0
919.859976	15000	1000	30000	30000	8000	0	0	0	8	1	0	0
920.863278	1000	23000	30000	30000	0	8000	0	0	1	7	0	0
921.866640	16000	23000	30000	30000	8000	8000	0	0	7	8	0	0
922.869970	16000	23000	30000	30000	7000	8000	0	0	8	8	0	0
923.873303	11000	20000	30000	30000	8000	8000	0	0	7	8	0	0
924.876642	13000	20000	30000	30000	8000	8000	0	0	8	8	0	0
925.879982	1000	1000	30000	30000	0	0	0	0	1	1	0	0
926.883290	17000	20000	30000	30000	7000	8000	0	0	7	7	0	0
927.886645	11000	20000	30000	30000	8000	8000	0	0	7	8	0	0
928.889927	13000	20000	30000	30000	8000	8000	0	0	8	8	0	0
929.893290	14000	20000	30000	30000	7000	8000	0	0	8	8	0	0
930.896634	1000	22000	30000	30000	0	7000	0	0	1	8	0	0
931.899962	15000	1000	30000	30000	8000	0	0	0	6	1	0	0
932.903259	15000	23000	30000	30000	8000	7000	0	0	8	6	0	0
933.906630	15000	23000	30000	30000	8000	8000	0	0	8	7	0	0
934.909963	15000	23000	30000	30000	8000	8000	0	0	8	8	0	0
935.913297	15000	23000	30000	30000	8000	8000	0	0	8	8	0	0
936.916631	1000	23000	30000	30000	0	8000	0	0	1	8	0	0
937.919966	15000	1000	30000	30000	8000	0	0	0	7	1	0	0
938.923298	15000	24000	30000	30000	8000	8000	0	0	8	7	0	0
939.926628	10000	21000	30000	30000	7000	8000	0	0	8	8	0	0
940.929960	10000	23000	30000	30000	7000	8000	0	0	7	8	0	0

941.933297	0	24000	30000	30000	0	8000	0	0	0	8	0	0
942.936623	11000	24000	30000	30000	8000	8000	0	0	7	8	0	0
943.939959	13000	1000	30000	30000	8000	0	0	0	8	1	0	0
944.943309	13000	25000	30000	30000	8000	8000	0	0	8	7	0	0
945.946632	13000	25000	30000	30000	8000	8000	0	0	8	8	0	0
946.949967	1000	25000	30000	30000	0	8000	0	0	1	8	0	0
947.953307	13000	25000	30000	30000	10000	8000	0	0	9	8	0	0
948.956633	0	25000	30000	30000	0	8000	0	0	0	8	0	0
949.961637	1000	1000	30000	30000	0	0	0	0	1	1	0	0
950.963304	10000	29000	30000	30000	8000	8000	0	0	7	7	0	0
951.966636	10000	21000	30000	30000	7000	8000	0	0	8	8	0	0
952.969935	10000	23000	30000	30000	7000	8000	0	0	7	8	0	0
953.973279	1000	24000	30000	30000	0	7000	0	0	1	8	0	0
954.976609	11000	1000	30000	30000	7000	0	0	0	6	1	0	0
955.979955	11000	25000	30000	30000	8000	8000	0	0	7	6	0	0
956.983304	11000	27000	30000	30000	8000	8000	0	0	8	8	0	0
957.986611	11000	27000	30000	30000	8000	8000	0	0	8	8	0	0
958.989942	1000	27000	30000	30000	0	8000	0	0	1	8	0	0
959.993269	11000	27000	30000	30000	7000	8000	0	0	7	8	0	0
960.996635	11000	1000	30000	30000	7000	0	0	0	7	1	0	0
961.999977	11000	28000	0	0	8000	8000	0	0	7	7	0	0
963.003267	11000	31000	0	0	8000	8000	0	0	8	8	0	0
964.006634	1000	34000	0	0	0	6000	0	0	1	8	0	0
965.009961	13000	3000	0	0	8000	0	0	0	7	3	0	0
966.013301	18000	3000	0	0	8000	0	0	0	8	3	0	0
967.016619	22000	0	0	0	8000	0	0	0	8	0	0	0
968.019976	26000	0	0	0	0	0	0	0	8	0	0	0
969.023300	0	0	0	0	0	0	0	0	0	0	0	0
970.026633	0	0	0	0	0	0	0	0	0	0	0	0
971.029960	0	0	0	0	0	0	0	0	0	0	0	0
972.033276	0	0	0	0	0	0	0	0	0	0	0	0
973.036594	0	0	0	0	0	0	0	0	0	0	0	0
974.039975	0	0	0	0	0	0	0	0	0	0	0	0
975.043290	0	0	0	0	0	0	0	0	0	0	0	0
976.046645	0	0	0	0	0	0	0	0	0	0	0	0
977.049962	0	0	0	0	0	0	0	0	0	0	0	0
978.053266	0	0	0	0	0	0	0	0	0	0	0	0
979.056651	0	0	0	0	0	0	0	0	0	0	0	0
980.059965	0	0	0	0	0	0	0	0	0	0	0	0
981.063309	0	0	0	0	0	0	0	0	0	0	0	0
982.066633	0	0	0	0	0	0	0	0	0	0	0	0
983.069978	0	0	0	0	0	0	0	0	0	0	0	0
984.073285	0	0	0	0	0	0	0	0	0	0	0	0
985.076630	0	0	0	0	0	0	0	0	0	0	0	0
986.079964	0	0	0	0	0	0	0	0	0	0	0	0

Table A.4: Server Receiving Data - Client Outgoing Bandwidth Limited

	Bandwidth (bytes per second)				Bytes Dropped				Retransmitted Packets			
Time Stamp	Group 0	Group 1	Group 2	Group 3	Group 0	Group 1	Group 2	Group 3	Group 0	Group 1	Group 2	Group 3
327.706693	0	0	0	0	0	0	0	0	0	0	0	0
328.710018	0	0	0	0	0	0	0	0	0	0	0	0
329.713366	0	0	0	0	0	0	0	0	0	0	0	0
330.716691	0	0	0	0	0	0	0	0	0	0	0	0
331.719994	0	0	0	0	0	0	0	0	0	0	0	0
332.723359	0	0	0	0	0	0	0	0	0	0	0	0
333.726687	0	0	0	0	0	0	0	0	0	0	0	0
334.730024	0	0	0	0	0	0	0	0	0	0	0	0
335.733350	0	0	0	0	0	0	0	0	0	0	0	0
336.736696	0	0	0	0	0	0	0	0	0	0	0	0
337.740020	0	0	0	0	0	0	0	0	0	0	0	0
338.743357	0	0	0	0	0	0	0	0	0	0	0	0
339.746689	0	0	0	0	0	0	0	0	0	0	0	0
340.750020	0	0	0	0	0	0	0	0	0	0	0	0
341.753356	0	0	0	0	0	0	0	0	0	0	0	0
342.756719	0	0	0	0	0	0	0	0	0	0	0	0
343.760044	0	0	0	0	0	0	0	0	0	0	0	0
344.763377	0	0	0	0	0	0	0	0	0	0	0	0
345.766712	0	0	0	0	0	0	0	0	0	0	0	0
346.770002	0	0	0	0	0	0	0	0	0	0	0	0
347.773368	0	0	0	0	0	0	0	0	0	0	0	0
348.776715	0	0	0	0	0	0	0	0	0	0	0	0
349.780045	0	0	0	0	0	0	0	0	0	0	0	0
350.783387	0	0	0	0	0	0	0	0	0	0	0	0
351.786678	0	0	0	0	0	0	0	0	0	0	0	0
352.790044	0	0	0	0	0	0	0	0	0	0	0	0
353.793354	0	0	0	0	0	0	0	0	0	0	0	0
354.796692	0	0	0	0	0	0	0	0	0	0	0	0
355.800023	0	0	0	0	0	0	0	0	0	0	0	0
356.803352	0	0	0	0	0	0	0	0	0	0	0	0
357.806690	0	0	0	0	0	0	0	0	0	0	0	0
358.810021	0	0	0	0	0	0	0	0	0	0	0	0
359.813359	0	0	0	0	0	0	0	0	0	0	0	0
360.816688	0	0	0	0	0	0	0	0	0	0	0	0
361.820026	0	0	0	0	0	0	0	0	0	0	0	0
362.823350	0	0	0	0	0	0	0	0	0	0	0	0
363.826699	0	0	0	0	0	0	0	0	0	0	0	0
364.830023	0	0	0	0	0	0	0	0	0	0	0	0
365.833354	0	0	0	0	0	0	0	0	0	0	0	0
366.836681	0	0	0	0	0	0	0	0	0	0	0	0

367.840015	0	0	0	0	0	0	0	0	0	0	0	0
368.843348	0	0	0	0	0	0	0	0	0	0	0	0
369.846693	0	0	0	0	0	0	0	0	0	0	0	0
370.850015	0	0	0	0	0	0	0	0	0	0	0	0
371.853356	0	0	0	0	0	0	0	0	0	0	0	0
372.856664	0	0	0	0	0	0	0	0	0	0	0	0
373.860006	0	0	0	0	0	0	0	0	0	0	0	0
374.863357	0	0	0	0	0	0	0	0	0	0	0	0
375.866694	0	0	0	0	0	0	0	0	0	0	0	0
376.870018	0	0	0	0	0	0	0	0	0	0	0	0
377.873368	0	0	0	0	0	0	0	0	0	0	0	0
378.876679	0	0	0	0	0	0	0	0	0	0	0	0
379.880025	0	0	0	0	0	0	0	0	0	0	0	0
380.883376	0	0	0	0	0	0	0	0	0	0	0	0
381.886678	0	0	0	0	0	0	0	0	0	0	0	0
382.890002	0	0	0	0	0	0	0	0	0	0	0	0
383.893352	0	0	0	0	0	0	0	0	0	0	0	0
384.896676	0	0	0	0	0	0	0	0	0	0	0	0
385.900002	0	0	0	0	0	0	0	0	0	0	0	0
386.903355	0	0	0	0	0	0	0	0	0	0	0	0
387.906691	0	0	0	0	0	0	0	0	0	0	0	0
388.910010	0	0	0	0	0	0	0	0	0	0	0	0
389.913352	0	0	0	0	0	0	0	0	0	0	0	0
390.916683	0	0	0	0	0	0	0	0	0	0	0	0
391.920014	0	0	0	0	0	0	0	0	0	0	0	0
392.923349	0	0	0	0	0	0	0	0	0	0	0	0
393.926652	0	0	0	0	0	0	0	0	0	0	0	0
394.930012	0	0	0	0	0	0	0	0	0	0	0	0
395.933347	0	0	0	0	0	0	0	0	0	0	0	0
396.936682	0	0	0	0	0	0	0	0	0	0	0	0
397.940015	0	0	0	0	0	0	0	0	0	0	0	0
398.943349	0	0	0	0	0	0	0	0	0	0	0	0
399.946677	0	0	0	0	0	0	0	0	0	0	0	0
400.950019	0	0	0	0	0	0	0	0	0	0	0	0
401.953345	0	0	0	0	0	0	0	0	0	0	0	0
402.956679	0	0	0	0	0	0	0	0	0	0	0	0
403.959987	10000	10000	10000	10000	0	0	0	0	0	0	0	0
404.963355	10000	10000	10000	10000	0	0	0	0	0	0	0	0
405.966689	10000	10000	10000	10000	0	0	0	0	0	0	0	0
406.970001	10000	10000	10000	10000	0	0	0	0	0	0	0	0
407.973310	10000	10000	10000	10000	0	0	0	0	0	0	0	0
408.976692	10000	10000	10000	10000	0	0	0	0	0	0	0	0
409.980025	10000	10000	10000	10000	0	0	0	0	0	0	0	0
410.983354	10000	10000	10000	10000	0	0	0	0	0	0	0	0
411.986675	10000	10000	10000	10000	0	0	0	0	0	0	0	0
412.990008	10000	10000	10000	10000	0	0	0	0	0	0	0	0
413.993348	10000	10000	10000	10000	0	0	0	0	0	0	0	0
414.996663	10000	10000	10000	10000	0	0	0	0	0	0	0	0

416.000014	10000	10000	10000	10000	0	0	0	0	0	0	0	0
417.003347	10000	10000	10000	10000	0	0	0	0	0	0	0	0
418.006678	10000	10000	10000	10000	0	0	0	0	0	0	0	0
419.010023	10000	10000	10000	10000	0	0	0	0	0	0	0	0
420.013355	10000	10000	10000	10000	0	0	0	0	0	0	0	0
421.016684	10000	10000	10000	10000	0	0	0	0	0	0	0	0
422.020013	10000	10000	10000	10000	0	0	0	0	0	0	0	0
423.023317	10000	10000	10000	10000	0	0	0	0	0	0	0	0
424.026683	10000	10000	10000	10000	0	0	0	0	0	0	0	0
425.029972	10000	10000	10000	10000	0	0	0	0	0	0	0	0
426.033353	10000	10000	10000	10000	0	0	0	0	0	0	0	0
427.036687	10000	10000	10000	10000	0	0	0	0	0	0	0	0
428.040013	10000	10000	10000	10000	0	0	0	0	0	0	0	0
429.043326	10000	10000	10000	10000	0	0	0	0	0	0	0	0
430.046685	10000	10000	10000	10000	0	0	0	0	0	0	0	0
431.050006	10000	10000	10000	10000	0	0	0	0	0	0	0	0
432.053357	10000	10000	10000	10000	0	0	0	0	0	0	0	0
433.056685	10000	10000	10000	10000	0	0	0	0	0	0	0	0
434.060012	15000	10000	10000	10000	0	0	0	0	0	0	0	0
435.063339	15000	10000	10000	10000	0	0	0	0	0	0	0	0
436.066696	15000	10000	10000	10000	0	0	0	0	0	0	0	0
437.070018	15000	10000	10000	10000	0	0	0	0	0	0	0	0
438.073335	15000	10000	10000	10000	0	0	0	0	0	0	0	0
439.076674	15000	10000	10000	10000	0	0	0	0	0	0	0	0
440.080021	15000	10000	10000	10000	0	0	0	0	0	0	0	0
441.083308	15000	10000	10000	10000	0	0	0	0	0	0	0	0
442.086638	15000	10000	10000	10000	0	0	0	0	0	0	0	0
443.089995	15000	10000	10000	10000	0	0	0	0	0	0	0	0
444.093338	15000	10000	10000	10000	0	0	0	0	0	0	0	0
445.096661	15000	10000	10000	10000	0	0	0	0	0	0	0	0
446.100026	15000	10000	10000	10000	0	0	0	0	0	0	0	0
447.103347	15000	10000	10000	10000	0	0	0	0	0	0	0	0
448.106648	15000	10000	10000	10000	0	0	0	0	0	0	0	0
449.109977	15000	10000	10000	10000	0	0	0	0	0	0	0	0
450.113344	15000	10000	10000	10000	0	0	0	0	0	0	0	0
451.116676	15000	10000	10000	10000	0	0	0	0	0	0	0	0
452.120012	15000	10000	10000	10000	0	0	0	0	0	0	0	0
453.123340	15000	10000	10000	10000	0	0	0	0	0	0	0	0
454.126676	15000	10000	10000	10000	0	0	0	0	0	0	0	0
455.130014	15000	10000	10000	10000	0	0	0	0	0	0	0	0
456.133348	15000	10000	10000	10000	0	0	0	0	0	0	0	0
457.136674	15000	10000	10000	10000	0	0	0	0	0	0	0	0
458.139978	15000	10000	10000	10000	0	0	0	0	0	0	0	0
459.143342	15000	10000	10000	10000	0	0	0	0	0	0	0	0
460.146679	15000	10000	10000	10000	0	0	0	0	0	0	0	0
461.150006	15000	10000	10000	10000	0	0	0	0	0	0	0	0
462.153347	15000	10000	10000	10000	0	0	0	0	0	0	0	0
463.156683	15000	10000	10000	10000	0	0	0	0	0	0	0	0

464.160016	20000	10000	10000	10000	0	0	0	0	0	0	0	0
465.163346	20000	10000	10000	10000	0	0	0	0	0	0	0	0
466.166683	20000	10000	10000	10000	0	0	0	0	0	0	0	0
467.170002	20000	10000	10000	10000	0	0	0	0	0	0	0	0
468.173349	20000	10000	10000	10000	0	0	0	0	0	0	0	0
469.176671	20000	10000	10000	10000	0	0	0	0	0	0	0	0
470.180002	20000	10000	10000	10000	0	0	0	0	0	0	0	0
471.183341	20000	10000	10000	10000	0	0	0	0	0	0	0	0
472.186675	20000	10000	10000	10000	0	0	0	0	0	0	0	0
473.190012	20000	10000	10000	10000	0	0	0	0	0	0	0	0
474.193340	20000	10000	10000	10000	0	0	0	0	0	0	0	0
475.196657	20000	10000	10000	10000	0	0	0	0	0	0	0	0
476.200021	20000	10000	10000	10000	0	0	0	0	0	0	0	0
477.203346	20000	10000	10000	10000	0	0	0	0	0	0	0	0
478.206662	20000	10000	10000	10000	0	0	0	0	0	0	0	0
479.210004	20000	10000	10000	10000	0	0	0	0	0	0	0	0
480.213336	20000	10000	10000	10000	0	0	0	0	0	0	0	0
481.216667	20000	10000	10000	10000	0	0	0	0	0	0	0	0
482.220025	20000	10000	10000	10000	0	0	0	0	0	0	0	0
483.223321	20000	10000	10000	10000	0	0	0	0	0	0	0	0
484.226673	20000	10000	10000	10000	0	0	0	0	0	0	0	0
485.230021	20000	10000	10000	10000	0	0	0	0	0	0	0	0
486.233342	20000	10000	10000	10000	0	0	0	0	0	0	0	0
487.236654	20000	10000	10000	10000	0	0	0	0	0	0	0	0
488.239997	20000	10000	10000	10000	0	0	0	0	0	0	0	0
489.243328	20000	10000	10000	10000	0	0	0	0	0	0	0	0
490.246655	20000	10000	10000	10000	0	0	0	0	0	0	0	0
491.249988	20000	10000	10000	10000	0	0	0	0	0	0	0	0
492.253330	20000	10000	10000	10000	0	0	0	0	0	0	0	0
493.256676	20000	10000	10000	10000	0	0	0	0	0	0	0	0
494.260101	25000	10000	10000	10000	0	0	0	0	0	0	0	0
495.263342	25000	10000	10000	10000	0	0	0	0	0	0	0	0
496.266707	25000	10000	10000	10000	0	0	0	0	0	0	0	0
497.270003	25000	10000	10000	10000	0	0	0	0	0	0	0	0
498.273319	25000	10000	10000	10000	0	0	0	0	0	0	0	0
499.276683	25000	10000	10000	10000	0	0	0	0	0	0	0	0
500.280024	25000	10000	10000	10000	0	0	0	0	0	0	0	0
501.283333	25000	10000	10000	10000	0	0	0	0	0	0	0	0
502.286665	25000	10000	10000	10000	0	0	0	0	0	0	0	0
503.289993	25000	10000	10000	10000	0	0	0	0	0	0	0	0
504.293338	25000	10000	10000	10000	0	0	0	0	0	0	0	0
505.296671	25000	10000	10000	10000	0	0	0	0	0	0	0	0
506.300005	25000	10000	10000	10000	0	0	0	0	0	0	0	0
507.303338	25000	10000	10000	10000	0	0	0	0	0	0	0	0
508.306684	25000	10000	10000	10000	0	0	0	0	0	0	0	0
509.309994	25000	10000	10000	10000	0	0	0	0	0	0	0	0
510.313338	25000	10000	10000	10000	0	0	0	0	0	0	0	0
511.316671	25000	10000	10000	10000	0	0	0	0	0	0	0	0

512.320005	25000	10000	10000	10000	0	0	0	0	0	0	0	0
513.323336	25000	10000	10000	10000	0	0	0	0	0	0	0	0
514.326670	25000	10000	10000	10000	0	0	0	0	0	0	0	0
515.330004	25000	10000	10000	10000	0	0	0	0	0	0	0	0
516.333342	25000	10000	10000	10000	0	0	0	0	0	0	0	0
517.336673	25000	10000	10000	10000	0	0	0	0	0	0	0	0
518.340011	25000	10000	10000	10000	0	0	0	0	0	0	0	0
519.343341	25000	10000	10000	10000	0	0	0	0	0	0	0	0
520.346675	25000	10000	10000	10000	0	0	0	0	0	0	0	0
521.349966	25000	10000	10000	10000	0	0	0	0	0	0	0	0
522.353342	25000	10000	10000	10000	0	0	0	0	0	0	0	0
523.356678	25000	10000	10000	10000	0	0	0	0	0	0	0	0
524.360014	30000	10000	10000	10000	0	0	0	0	0	0	0	0
525.363331	30000	10000	10000	10000	0	0	0	0	0	0	0	0
526.366665	30000	10000	10000	10000	0	0	0	0	0	0	0	0
527.370000	30000	10000	10000	10000	0	0	0	0	0	0	0	0
528.373345	30000	10000	10000	10000	0	0	0	0	0	0	0	0
529.376660	30000	10000	10000	10000	0	0	0	0	0	0	0	0
530.380012	30000	10000	10000	10000	0	0	0	0	0	0	0	0
531.383334	30000	10000	10000	10000	0	0	0	0	0	0	0	0
532.386677	30000	10000	10000	10000	0	0	0	0	0	0	0	0
533.390018	30000	10000	10000	10000	0	0	0	0	0	0	0	0
534.393351	30000	10000	10000	10000	0	0	0	0	0	0	0	0
535.396682	30000	10000	10000	10000	0	0	0	0	0	0	0	0
536.400016	30000	10000	10000	10000	0	0	0	0	0	0	0	0
537.403339	30000	10000	10000	10000	0	0	0	0	0	0	0	0
538.406697	30000	10000	10000	10000	0	0	0	0	0	0	0	0
539.409989	30000	10000	10000	10000	0	0	0	0	0	0	0	0
540.413329	30000	10000	10000	10000	0	0	0	0	0	0	0	0
541.416662	30000	10000	10000	10000	0	0	0	0	0	0	0	0
542.419998	30000	10000	10000	10000	0	0	0	0	0	0	0	0
543.423445	30000	10000	14000	10000	0	0	0	0	0	0	0	0
544.426625	30000	10000	13000	13000	0	0	0	0	0	0	0	0
545.429959	32000	10000	12000	16000	0	0	0	0	0	0	0	0
546.433320	40000	10000	10000	10000	0	0	0	0	0	0	0	0
547.436624	40000	10000	10000	10000	0	0	0	0	0	0	0	0
548.439959	37000	10000	10000	10000	0	0	0	0	0	0	0	0
549.443294	30000	10000	10000	10000	0	0	0	0	0	0	0	0
550.446626	30000	10000	10000	10000	0	0	0	0	0	0	0	0
551.449968	30000	10000	10000	10000	0	0	0	0	0	0	0	0
552.453297	31000	10000	10000	10000	0	0	0	0	0	0	0	0
553.456631	29000	10000	10000	10000	0	0	0	0	0	0	0	0
554.459959	30000	15000	10000	10000	0	0	0	0	0	0	0	0
555.463301	30000	15000	10000	10000	0	0	0	0	0	0	0	0
556.466625	30000	15000	10000	10000	0	0	0	0	0	0	0	0
557.469964	30000	15000	10000	10000	0	0	0	0	0	0	0	0
558.473297	30000	15000	10000	10000	0	0	0	0	0	0	0	0
559.476624	30000	15000	10000	10000	0	0	0	0	0	0	0	0

560.479959	30000	15000	10000	10000	0	0	0	0	0	0	0	0
561.483341	30000	15000	10000	10000	0	0	0	0	0	0	0	0
562.486689	30000	15000	10000	10000	0	0	0	0	0	0	0	0
563.489990	30000	15000	10000	10000	0	0	0	0	0	0	0	0
564.493326	30000	15000	10000	10000	0	0	0	0	0	0	0	0
565.496655	30000	15000	10000	10000	0	0	0	0	0	0	0	0
566.499991	30000	15000	10000	10000	0	0	0	0	0	0	0	0
567.503326	30000	15000	10000	10000	0	0	0	0	0	0	0	0
568.506668	30000	15000	10000	10000	0	0	0	0	0	0	0	0
569.509960	31000	15000	11000	10000	0	0	0	0	0	0	0	0
570.513314	29000	15000	9000	10000	0	0	0	0	0	0	0	0
571.516666	30000	15000	10000	10000	0	0	0	0	0	0	0	0
572.519998	30000	15000	10000	10000	0	0	0	0	0	0	0	0
573.523302	30000	15000	10000	10000	0	0	0	0	0	0	0	0
574.527289	30000	29000	10000	10000	0	0	0	0	0	0	0	0
575.529955	30000	16000	10000	10000	0	0	0	0	0	0	0	0
576.533290	30000	15000	10000	10000	0	0	0	0	0	0	0	0
577.536625	30000	15000	10000	10000	0	0	0	0	0	0	0	0
578.539956	30000	15000	10000	10000	0	0	0	0	0	0	0	0
579.543293	30000	15000	10000	10000	0	0	0	0	0	0	0	0
580.546622	30000	15000	10000	10000	0	0	0	0	0	0	0	0
581.549959	30000	15000	10000	10000	0	0	0	0	0	0	0	0
582.553288	30000	15000	10000	10000	0	0	0	0	0	0	0	0
583.556623	30000	19000	10000	10000	0	0	0	0	0	0	0	0
584.559953	30000	20000	10000	10000	0	0	0	0	0	0	0	0
585.563288	30000	20000	10000	10000	0	0	0	0	0	0	0	0
586.566625	30000	20000	10000	10000	0	0	0	0	0	0	0	0
587.569963	31000	20000	10000	10000	0	0	0	0	0	0	0	0
588.573304	29000	20000	10000	10000	0	0	0	0	0	0	0	0
589.576641	30000	20000	10000	10000	0	0	0	0	0	0	0	0
590.579988	30000	20000	10000	10000	0	0	0	0	0	0	0	0
591.583290	30000	21000	10000	10000	0	0	0	0	0	0	0	0
592.586650	30000	19000	10000	10000	0	0	0	0	0	0	0	0
593.589984	30000	20000	10000	10000	0	0	0	0	0	0	0	0
594.593327	30000	20000	10000	10000	0	0	0	0	0	0	0	0
595.596653	30000	20000	10000	10000	0	0	0	0	0	0	0	0
596.599987	30000	20000	10000	10000	0	0	0	0	0	0	0	0
597.603298	30000	20000	10000	10000	0	0	0	0	0	0	0	0
598.606637	30000	20000	10000	10000	0	0	0	0	0	0	0	0
599.610003	30000	20000	10000	10000	0	0	0	0	0	0	0	0
600.613314	30000	20000	10000	10000	0	0	0	0	0	0	0	0
601.616674	30000	20000	10000	10000	0	0	0	0	0	0	0	0
602.619957	30000	20000	10000	10000	0	0	0	0	0	0	0	0
603.623333	30000	20000	10000	10000	0	0	0	0	0	0	0	0
604.626673	31000	20000	10000	10000	0	0	0	0	0	0	0	0
605.629960	29000	20000	10000	10000	0	0	0	0	0	0	0	0
606.633367	30000	20000	11000	10000	0	0	0	0	0	0	0	0
607.636620	30000	20000	10000	11000	0	0	0	0	0	0	0	0

608.640134	30000	20000	10000	9000	0	0	0	0	0	0	0	0
609.643289	30000	20000	10000	11000	0	0	0	0	0	0	0	0
610.646619	31000	20000	10000	10000	0	0	0	0	0	0	0	0
611.649951	30000	20000	10000	10000	0	0	0	0	0	0	0	0
612.653285	30000	20000	10000	10000	0	0	0	0	0	0	0	0
613.656619	30000	26000	10000	10000	0	0	0	0	0	0	0	0
614.659954	30000	25000	10000	10000	0	0	0	0	0	0	0	0
615.663287	30000	25000	10000	10000	0	0	0	0	0	0	0	0
616.666629	30000	25000	10000	10000	0	0	0	0	0	0	0	0
617.669970	30000	25000	10000	10000	0	0	0	0	0	0	0	0
618.673296	30000	25000	10000	10000	0	0	0	0	0	0	0	0
619.676668	30000	25000	10000	10000	0	0	0	0	0	0	0	0
620.679963	30000	25000	10000	10000	0	0	0	0	0	0	0	0
621.683317	30000	25000	10000	10000	0	0	0	0	0	0	0	0
622.686666	30000	24000	10000	10000	0	0	0	0	0	0	0	0
623.689988	30000	26000	10000	10000	0	0	0	0	0	0	0	0
624.693301	30000	25000	10000	10000	0	0	0	0	0	0	0	0
625.696666	30000	25000	10000	10000	0	0	0	0	0	0	0	0
626.699986	30000	25000	10000	10000	0	0	0	0	0	0	0	0
627.703285	30000	25000	10000	10000	0	0	0	0	0	0	0	0
628.706671	30000	25000	10000	10000	0	0	0	0	0	0	0	0
629.709987	30000	25000	10000	10000	0	0	0	0	0	0	0	0
630.713324	30000	25000	10000	10000	0	0	0	0	0	0	0	0
631.716658	30000	25000	10000	10000	0	0	0	0	0	0	0	0
632.719989	30000	25000	10000	10000	0	0	0	0	0	0	0	0
633.723324	30000	25000	10000	10000	0	0	0	0	0	0	0	0
634.726658	30000	25000	10000	10000	0	0	0	0	0	0	0	0
635.729990	30000	25000	10000	10000	0	0	0	0	0	0	0	0
636.733328	30000	25000	10000	10000	0	0	0	0	0	0	0	0
637.736934	30000	25000	10000	10000	0	0	0	0	0	0	0	0
638.739980	30000	25000	10000	10000	0	0	0	0	0	0	0	0
639.743309	30000	25000	10000	10000	0	0	0	0	0	0	0	0
640.746642	30000	25000	10000	10000	0	0	0	0	0	0	0	0
641.749975	30000	25000	10000	10000	0	0	0	0	0	0	0	0
642.753305	30000	25000	10000	10000	0	0	0	0	0	0	0	0
643.756636	30000	30000	10000	10000	0	0	0	0	0	0	0	0
644.759948	30000	30000	10000	10000	0	0	0	0	0	0	0	0
645.763299	30000	30000	10000	10000	0	0	0	0	0	0	0	0
646.766654	30000	30000	10000	10000	0	0	0	0	0	0	0	0
647.769994	30000	30000	10000	10000	0	0	0	0	0	0	0	0
648.773321	30000	30000	10000	10000	0	0	0	0	0	0	0	0
649.776660	30000	30000	10000	10000	0	0	0	0	0	0	0	0
650.780008	30000	30000	10000	10000	0	0	0	0	0	0	0	0
651.783340	30000	30000	10000	10000	0	0	0	0	0	0	0	0
652.786671	30000	30000	10000	10000	0	0	0	0	0	0	0	0
653.790028	30000	30000	10000	10000	0	0	0	0	0	0	0	0
654.793323	30000	30000	10000	10000	0	0	0	0	0	0	0	0
655.796656	30000	30000	10000	10000	0	0	0	0	0	0	0	0

656.799991	30000	30000	10000	10000	0	0	0	0	0	0	0	0
657.803325	30000	30000	10000	10000	0	0	0	0	0	0	0	0
658.806657	30000	30000	10000	10000	0	0	0	0	0	0	0	0
659.809988	30000	30000	10000	10000	0	0	0	0	0	0	0	0
660.813325	30000	30000	10000	10000	0	0	0	0	0	0	0	0
661.816625	30000	30000	10000	10000	0	0	0	0	0	0	0	0
662.819990	30000	30000	10000	10000	0	0	0	0	0	0	0	0
663.823324	30000	30000	10000	10000	0	0	0	0	0	0	0	0
664.826661	30000	30000	10000	10000	0	0	0	0	0	0	0	0
665.829989	30000	30000	10000	10000	0	0	0	0	0	0	0	0
666.833325	30000	30000	10000	10000	0	0	0	0	0	0	0	0
667.836654	30000	30000	10000	10000	0	0	0	0	0	0	0	0
668.840243	30000	30000	10000	10000	0	0	0	0	0	0	0	0
669.843526	30000	30000	10000	10000	0	0	0	0	0	0	0	0
670.846636	30000	30000	10000	10000	0	0	0	0	0	0	0	0
671.849968	30000	30000	10000	10000	0	0	0	0	0	0	0	0
672.853299	30000	30000	10000	10000	0	0	0	0	0	0	0	0
673.856643	30000	30000	15000	10000	0	0	0	0	0	0	0	0
674.859971	30000	30000	15000	10000	0	0	0	0	0	0	0	0
675.863305	30000	30000	15000	10000	0	0	0	0	0	0	0	0
676.866658	30000	30000	15000	10000	0	0	0	0	0	0	0	0
677.869953	30000	30000	15000	10000	0	0	0	0	0	0	0	0
678.873302	30000	30000	15000	10000	0	0	0	0	0	0	0	0
679.876636	13000	20000	15000	10000	0	0	0	0	0	0	0	0
680.879946	13000	23000	15000	10000	0	0	0	0	13	9	0	0
681.883316	0	26000	15000	10000	0	0	0	0	0	12	0	0
682.886653	1000	0	15000	10000	0	0	0	0	1	0	0	0
683.889987	14000	26000	15000	10000	0	0	0	0	14	11	0	0
684.893329	0	34000	15000	10000	0	0	0	0	0	11	0	0
685.896654	0	38000	15000	10000	0	0	0	0	0	11	0	0
686.899981	0	1000	15000	10000	0	0	0	0	0	1	0	0
687.903293	11000	41000	15000	10000	0	0	0	0	8	11	0	0
688.906650	11000	45000	15000	10000	0	0	0	0	8	12	0	0
689.909981	15000	76000	15000	10000	0	0	0	0	8	12	0	0
690.913288	17000	30000	15000	10000	0	0	0	0	8	0	0	0
691.916651	1000	30000	15000	10000	0	0	0	0	1	0	0	0
692.919987	19000	30000	15000	10000	0	0	0	0	8	0	0	0
693.923303	23000	30000	15000	10000	0	0	0	0	9	0	0	0
694.926621	26000	30000	15000	10000	0	0	0	0	9	0	0	0
695.929987	34000	30000	15000	10000	0	0	0	0	11	0	0	0
696.933294	36000	30000	15000	10000	0	0	0	0	11	0	0	0
697.936654	1000	30000	15000	10000	0	0	0	0	1	0	0	0
698.939984	38000	30000	15000	10000	0	0	0	0	10	0	0	0
699.943316	42000	30000	15000	10000	0	0	0	0	12	0	0	0
700.946651	43000	23000	15000	10000	0	0	0	0	12	0	0	0
701.949985	11000	27000	15000	10000	0	0	0	0	11	6	0	0
702.953281	0	23000	15000	10000	0	0	0	0	0	7	0	0
703.956633	12000	1000	20000	10000	0	0	0	0	8	1	0	0

704.959980	0	22000	20000	10000	0	0	0	0	0	8	0	0
705.963323	15000	23000	20000	10000	0	0	0	0	8	9	0	0
706.966649	18000	27000	20000	10000	0	0	0	0	8	9	0	0
707.969986	22000	26000	20000	10000	0	0	0	0	9	11	0	0
708.973301	23000	26000	20000	10000	0	0	0	0	9	11	0	0
709.976612	24000	1000	20000	10000	0	0	0	0	9	1	0	0
710.979947	1000	25000	20000	10000	0	0	0	0	1	10	0	0
711.983316	25000	26000	20000	10000	0	0	0	0	8	11	0	0
712.986652	27000	26000	20000	10000	0	0	0	0	11	11	0	0
713.989986	32000	26000	20000	10000	0	0	0	0	11	11	0	0
714.993274	33000	25000	20000	10000	0	0	0	0	11	11	0	0
715.996611	34000	1000	20000	10000	0	0	0	0	11	1	0	0
716.999961	1000	24000	20000	10000	0	0	0	0	1	8	0	0
718.003313	35000	25000	20000	10000	0	0	0	0	10	9	0	0
719.006647	39000	26000	20000	10000	0	0	0	0	11	9	0	0
720.009984	41000	25000	20000	10000	0	0	0	0	12	11	0	0
721.013275	43000	24000	20000	10000	0	0	0	0	12	9	0	0
722.016653	1000	1000	20000	10000	0	0	0	0	1	1	0	0
723.019983	43000	23000	20000	10000	0	0	0	0	11	8	0	0
724.023286	46000	24000	20000	10000	0	0	0	0	12	9	0	0
725.026656	13000	20000	20000	10000	0	0	0	0	12	9	0	0
726.029989	21000	21000	20000	10000	0	0	0	0	8	9	0	0
727.033319	13000	20000	20000	10000	0	0	0	0	9	9	0	0
728.036648	1000	1000	20000	10000	0	0	0	0	1	1	0	0
729.039981	15000	19000	20000	10000	0	0	0	0	7	8	0	0
730.043290	19000	20000	20000	10000	0	0	0	0	8	9	0	0
731.046657	22000	20000	20000	10000	0	0	0	0	9	9	0	0
732.049991	30000	21000	20000	10000	0	0	0	0	9	9	0	0
733.053317	13000	20000	20000	10000	0	0	0	0	11	9	0	0
734.056653	1000	1000	25000	10000	0	0	0	0	1	1	0	0
735.060004	15000	19000	25000	10000	0	0	0	0	7	8	0	0
736.063314	19000	20000	25000	10000	0	0	0	0	8	9	0	0
737.066655	24000	21000	25000	10000	0	0	0	0	9	9	0	0
738.069976	13000	20000	25000	10000	0	0	0	0	9	9	0	0
739.073318	1000	1000	25000	10000	0	0	0	0	1	1	0	0
740.076656	15000	19000	25000	10000	0	0	0	0	7	8	0	0
741.079987	19000	20000	25000	10000	0	0	0	0	8	9	0	0
742.083282	13000	20000	25000	10000	0	0	0	0	9	9	0	0
743.086645	21000	21000	25000	10000	0	0	0	0	8	9	0	0
744.089977	13000	20000	25000	10000	0	0	0	0	9	9	0	0
745.093295	1000	1000	25000	10000	0	0	0	0	1	1	0	0
746.096665	15000	19000	25000	10000	0	0	0	0	7	8	0	0
747.099979	19000	20000	25000	10000	0	0	0	0	8	9	0	0
748.103312	20000	20000	25000	10000	0	0	0	0	9	9	0	0
749.106655	25000	21000	25000	10000	0	0	0	0	9	9	0	0
750.109975	13000	20000	25000	10000	0	0	0	0	9	9	0	0
751.113270	16000	1000	25000	10000	0	0	0	0	8	1	0	0
752.116647	0	19000	25000	10000	0	0	0	0	0	8	0	0

753.119978	19000	20000	25000	10000	0	0	0	0	8	9	0	0
754.123314	24000	21000	25000	10000	0	0	0	0	9	9	0	0
755.126643	13000	20000	25000	10000	0	0	0	0	9	9	0	0
756.129950	16000	0	25000	10000	0	0	0	0	8	0	0	0
757.133313	0	20000	25000	10000	0	0	0	0	0	9	0	0
758.136909	19000	20000	25000	10000	0	0	0	0	8	9	0	0
759.139980	20000	22000	25000	10000	0	0	0	0	9	9	0	0
760.143314	22000	21000	25000	10000	0	0	0	0	9	9	0	0
761.146616	23000	1000	25000	10000	0	0	0	0	9	1	0	0
762.149979	1000	20000	25000	10000	0	0	0	0	1	8	0	0
763.153310	23000	21000	25000	10000	0	0	0	0	8	9	0	0
764.156645	10000	41000	30000	10000	0	0	0	0	9	17	0	0
765.159983	12000	1000	30000	10000	0	0	0	0	8	1	0	0
766.163304	1000	20000	30000	10000	0	0	0	0	1	8	0	0
767.166643	12000	21000	30000	10000	0	0	0	0	7	9	0	0
768.169984	14000	21000	30000	10000	0	0	0	0	8	9	0	0
769.173308	16000	23000	30000	10000	0	0	0	0	8	9	0	0
770.176649	19000	22000	30000	10000	0	0	0	0	8	9	0	0
771.179941	19000	22000	30000	10000	0	0	0	0	9	9	0	0
772.183299	1000	1000	30000	10000	0	0	0	0	1	1	0	0
773.186654	20000	21000	30000	10000	0	0	0	0	8	8	0	0
774.189983	22000	22000	30000	10000	0	0	0	0	9	9	0	0
775.193315	24000	25000	30000	10000	0	0	0	0	9	9	0	0
776.196638	26000	24000	30000	10000	0	0	0	0	9	9	0	0
777.199955	26000	24000	30000	10000	0	0	0	0	11	9	0	0
778.203307	1000	1000	30000	10000	0	0	0	0	1	1	0	0
779.206647	27000	23000	30000	10000	0	0	0	0	10	8	0	0
780.209945	29000	24000	30000	10000	0	0	0	0	11	9	0	0
781.213296	31000	24000	30000	10000	0	0	0	0	11	9	0	0
782.216643	37000	23000	30000	10000	0	0	0	0	11	9	0	0
783.219956	37000	22000	30000	10000	0	0	0	0	11	9	0	0
784.223273	37000	1000	30000	10000	0	0	0	0	12	1	0	0
785.226647	1000	21000	30000	10000	0	0	0	0	1	8	0	0
786.229972	36000	22000	30000	10000	0	0	0	0	10	9	0	0
787.233310	37000	22000	30000	10000	0	0	0	0	11	8	0	0
788.236643	25000	20000	30000	10000	0	0	0	0	18	7	0	0
789.239985	1000	1000	30000	10000	0	0	0	0	1	1	0	0
790.243307	10000	19000	30000	10000	0	0	0	0	7	7	0	0
791.246641	13000	20000	30000	10000	0	0	0	0	8	8	0	0
792.249989	30000	20000	30000	10000	0	0	0	0	15	8	0	0
793.253310	16000	22000	30000	10000	0	0	0	0	8	8	0	0
794.256673	1000	22000	30000	15000	0	0	0	0	1	8	0	0
795.259992	15000	0	30000	15000	0	0	0	0	7	0	0	0
796.263285	16000	24000	30000	15000	0	0	0	0	8	7	0	0
797.266635	18000	25000	30000	15000	0	0	0	0	8	8	0	0
798.269935	17000	28000	30000	15000	0	0	0	0	9	8	0	0
799.273293	1000	29000	30000	15000	0	0	0	0	1	8	0	0
800.276624	16000	1000	30000	15000	0	0	0	0	8	1	0	0

801.279963	17000	29000	30000	15000	0	0	0	0	9	7	0	0
802.283318	17000	32000	30000	15000	0	0	0	0	9	7	0	0
803.286639	20000	33000	30000	15000	0	0	0	0	9	8	0	0
804.289995	19000	35000	30000	15000	0	0	0	0	9	8	0	0
805.293308	1000	34000	30000	15000	0	0	0	0	1	8	0	0
806.296641	18000	1000	30000	15000	0	0	0	0	8	1	0	0
807.299973	19000	33000	30000	15000	0	0	0	0	9	6	0	0
808.303312	19000	34000	30000	15000	0	0	0	0	8	7	0	0
809.306632	20000	34000	30000	15000	0	0	0	0	8	7	0	0
810.309968	19000	35000	30000	15000	0	0	0	0	8	8	0	0
811.313276	1000	34000	30000	15000	0	0	0	0	1	7	0	0
812.316888	18000	1000	30000	15000	0	0	0	0	7	1	0	0
813.319977	19000	33000	30000	15000	0	0	0	0	8	6	0	0
814.323304	19000	34000	30000	15000	0	0	0	0	8	7	0	0
815.326638	21000	34000	30000	15000	0	0	0	0	8	7	0	0
816.329978	20000	34000	30000	15000	0	0	0	0	8	8	0	0
817.333300	1000	33000	30000	15000	0	0	0	0	1	7	0	0
818.340412	20000	1000	30000	15000	0	0	0	0	8	1	0	0
819.343308	19000	32000	30000	15000	0	0	0	0	7	7	0	0
820.346640	20000	33000	30000	15000	0	0	0	0	8	8	0	0
821.349974	20000	33000	30000	15000	0	0	0	0	7	8	0	0
822.353301	19000	36000	30000	15000	0	0	0	0	8	8	0	0
823.356640	18000	35000	30000	15000	0	0	0	0	8	8	0	0
824.359937	0	21000	29000	17000	0	0	0	0	0	8	0	0
825.363290	10000	1000	30000	23000	0	0	0	0	7	1	1	3
826.366641	12000	21000	31000	20000	0	0	0	0	7	7	1	0
827.369971	1000	22000	30000	20000	0	0	0	0	1	8	0	0
828.373303	11000	24000	30000	20000	0	0	0	0	6	8	0	0
829.376604	12000	25000	30000	20000	0	0	0	0	7	8	0	0
830.379932	14000	25000	30000	20000	0	0	0	0	7	8	0	0
831.383286	1000	1000	30000	20000	0	0	0	0	1	1	0	0
832.386623	14000	24000	30000	20000	0	0	0	0	7	7	0	0
833.389932	17000	25000	30000	20000	0	0	0	0	8	8	0	0
834.393281	18000	25000	30000	20000	0	0	0	0	8	8	0	0
835.396626	23000	25000	30000	20000	0	0	0	0	8	8	0	0
836.399934	24000	24000	30000	20000	0	0	0	0	8	8	0	0
837.403287	1000	1000	30000	20000	0	0	0	0	1	1	0	0
838.406641	23000	23000	30000	20000	0	0	0	0	7	7	0	0
839.409962	24000	24000	30000	20000	0	0	0	0	8	7	0	0
840.413291	24000	24000	30000	20000	0	0	0	0	8	8	0	0
841.416628	27000	23000	30000	20000	0	0	0	0	7	8	0	0
842.419973	26000	22000	30000	20000	0	0	0	0	8	8	0	0
843.423302	1000	1000	30000	20000	0	0	0	0	1	1	0	0
844.426625	25000	21000	30000	20000	0	0	0	0	7	7	0	0
845.429961	26000	22000	30000	20000	0	0	0	0	8	8	0	0
846.433293	26000	22000	30000	20000	0	0	0	0	8	8	0	0
847.436638	28000	22000	30000	20000	0	0	0	0	8	8	0	0
848.439930	27000	21000	30000	20000	0	0	0	0	8	8	0	0

849.443305	1000	1000	30000	20000	0	0	0	0	1	1	0	0
850.446622	26000	20000	30000	20000	0	0	0	0	7	7	0	0
851.449964	27000	21000	30000	20000	0	0	0	0	8	8	0	0
852.453332	27000	21000	30000	20000	0	0	0	0	8	8	0	0
853.456638	27000	21000	30000	20000	0	0	0	0	8	8	0	0
854.459972	12000	20000	30000	25000	0	0	0	0	8	8	0	0
855.463302	1000	1000	30000	25000	0	0	0	0	1	1	0	0
856.467403	11000	19000	39000	25000	0	0	0	0	6	7	0	0
857.471383	12000	22000	48000	25000	0	0	0	0	7	8	0	0
858.473263	14000	23000	33000	25000	0	0	0	0	7	8	0	0
859.478668	1000	25000	27000	46000	0	0	0	0	1	8	0	0
860.479931	13000	26000	32000	28000	0	0	0	0	7	8	0	0
861.483262	14000	1000	31000	26000	0	0	0	0	8	1	0	0
862.486596	14000	26000	29000	25000	0	0	0	0	8	7	0	0
863.489931	16000	29000	30000	25000	0	0	0	0	8	8	0	0
864.493262	11000	29000	30000	25000	0	0	0	0	8	8	0	0
865.496593	11000	22000	30000	25000	0	0	0	0	8	8	0	0
866.499932	1000	23000	30000	25000	0	0	0	0	1	8	0	0
867.503260	10000	1000	31000	25000	0	0	0	0	7	1	0	0
868.506619	11000	23000	29000	25000	0	0	0	0	8	7	0	0
869.509925	13000	26000	31000	25000	0	0	0	0	8	8	0	0
870.513258	13000	26000	30000	25000	0	0	0	0	8	8	0	0
871.516595	13000	29000	29000	25000	0	0	0	0	8	7	0	0
872.519959	1000	30000	30000	25000	0	0	0	0	1	8	0	0
873.523273	12000	1000	31000	25000	0	0	0	0	7	1	0	0
874.526596	13000	30000	30000	25000	0	0	0	0	8	7	0	0
875.529925	10000	21000	30000	25000	0	0	0	0	8	8	0	0
876.533261	11000	23000	30000	25000	0	0	0	0	7	8	0	0
877.536593	1000	25000	30000	25000	0	0	0	0	1	8	0	0
878.539924	10000	25000	30000	25000	0	0	0	0	7	8	0	0
879.543275	11000	1000	30000	25000	0	0	0	0	8	1	0	0
880.546615	11000	26000	30000	25000	0	0	0	0	8	7	0	0
881.549927	14000	28000	29000	25000	0	0	0	0	8	8	0	0
882.553320	15000	28000	31000	25000	0	0	0	0	8	7	0	0
883.556591	1000	31000	29000	29000	0	0	0	0	1	8	0	0
884.559939	8000	21000	31000	31000	0	0	0	0	7	8	0	0
885.563626	10000	1000	30000	30000	0	0	0	0	8	1	0	0
886.566589	12000	23000	29000	30000	0	0	0	0	7	8	0	0
887.569938	1000	22000	30000	30000	0	0	0	0	1	7	0	0
888.573269	11000	23000	30000	29000	0	0	0	0	6	8	0	0
889.576619	14000	25000	30000	30000	0	0	0	0	7	8	0	0
890.579938	13000	25000	31000	31000	0	0	0	0	7	8	0	0
891.583264	1000	25000	29000	30000	0	0	0	0	1	8	0	0
892.586639	12000	1000	30000	29000	0	0	0	0	7	1	0	0
893.589956	13000	24000	30000	31000	0	0	0	0	8	7	0	0
894.593277	13000	25000	30000	29000	0	0	0	0	8	8	0	0
895.596625	12000	28000	30000	31000	0	0	0	0	8	8	0	0
896.599986	1000	0	31000	30000	0	0	0	0	1	0	0	0

897.603320	10000	0	30000	30000	0	0	0	0	6	0	0	0
898.606626	13000	26000	30000	29000	0	0	0	0	8	8	0	0
899.609961	13000	25000	29000	31000	0	0	0	0	8	8	0	0
900.613270	15000	1000	31000	29000	0	0	0	0	8	1	0	0
901.616620	11000	24000	29000	31000	0	0	0	0	8	7	0	0
902.619948	1000	20000	31000	30000	0	0	0	0	1	8	0	0
903.623303	12000	20000	30000	29000	0	0	0	0	7	8	0	0
904.626649	14000	22000	29000	30000	0	0	0	0	8	8	0	0
905.629985	14000	23000	30000	31000	0	0	0	0	8	8	0	0
906.633297	16000	1000	30000	30000	0	0	0	0	8	1	0	0
907.636633	15000	22000	30000	29000	0	0	0	0	8	7	0	0
908.639927	1000	23000	30000	31000	0	0	0	0	1	8	0	0
909.643274	14000	23000	31000	30000	0	0	0	0	7	8	0	0
910.646605	15000	24000	29000	29000	0	0	0	0	8	8	0	0
911.649939	15000	23000	31000	31000	0	0	0	0	8	7	0	0
912.653305	17000	1000	30000	30000	0	0	0	0	8	1	0	0
913.656626	11000	22000	30000	29000	0	0	0	0	8	7	0	0
914.659936	1000	20000	29000	31000	0	0	0	0	1	8	0	0
915.663295	12000	20000	31000	29000	0	0	0	0	6	8	0	0
916.666605	14000	22000	30000	30000	0	0	0	0	8	8	0	0
917.669932	14000	23000	30000	30000	0	0	0	0	8	8	0	0
918.673296	15000	1000	30000	30000	0	0	0	0	7	1	0	0
919.676613	1000	22000	30000	30000	0	0	0	0	1	7	0	0
920.680266	14000	23000	29000	30000	0	0	0	0	7	8	0	0
921.683319	15000	23000	31000	30000	0	0	0	0	8	8	0	0
922.686631	15000	24000	30000	30000	0	0	0	0	8	8	0	0
923.690055	17000	23000	30000	30000	0	0	0	0	8	8	0	0
924.693401	16000	1000	30000	30000	0	0	0	0	8	1	0	0
925.696602	0	22000	30000	31000	0	0	0	0	0	7	0	0
926.699942	11000	20000	30000	30000	0	0	0	0	7	8	0	0
927.703273	13000	20000	30000	30000	0	0	0	0	8	8	0	0
928.706634	18000	21000	30000	30000	0	0	0	0	8	8	0	0
929.709951	11000	20000	30000	30000	0	0	0	0	7	8	0	0
930.713293	1000	1000	30000	30000	0	0	0	0	1	1	0	0
931.716632	12000	19000	30000	30000	0	0	0	0	7	7	0	0
932.719963	14000	20000	30000	30000	0	0	0	0	8	8	0	0
933.723294	16000	22000	30000	30000	0	0	0	0	7	8	0	0
934.726627	15000	24000	30000	30000	0	0	0	0	8	7	0	0
935.729962	1000	23000	30000	30000	0	0	0	0	1	7	0	0
936.733294	14000	1000	30000	30000	0	0	0	0	7	1	0	0
937.736628	15000	22000	30000	30000	0	0	0	0	8	7	0	0
938.739920	15000	23000	30000	30000	0	0	0	0	8	8	0	0
939.743302	16000	23000	30000	30000	0	0	0	0	8	8	0	0
940.746633	15000	25000	30000	30000	0	0	0	0	8	8	0	0
941.749921	10000	21000	30000	30000	0	0	0	0	8	8	0	0
942.753296	0	1000	30000	30000	0	0	0	0	0	1	0	0
943.756587	10000	22000	30000	30000	0	0	0	0	7	7	0	0
944.759968	11000	24000	30000	30000	0	0	0	0	7	8	0	0

945.763254	13000	24000	30000	30000	0	0	0	0	8	8	0	0
946.766628	1000	26000	30000	30000	0	0	0	0	1	8	0	0
947.769959	12000	25000	30000	30000	0	0	0	0	7	8	0	0
948.773292	13000	1000	30000	30000	0	0	0	0	8	1	0	0
949.776618	14000	24000	30000	30000	0	0	0	0	8	7	0	0
950.779970	0	25000	30000	30000	0	0	0	0	0	8	0	0
951.783285	0	25000	30000	30000	0	0	0	0	0	8	0	0
952.786632	11000	30000	30000	30000	0	0	0	0	8	8	0	0
953.789927	10000	21000	30000	30000	0	0	0	0	8	8	0	0
954.793264	0	1000	30000	30000	0	0	0	0	0	1	0	0
955.796652	10000	22000	30000	30000	0	0	0	0	7	7	0	0
956.799958	12000	24000	30000	30000	0	0	0	0	7	8	0	0
957.803317	1000	26000	30000	30000	0	0	0	0	1	7	0	0
958.806645	10000	27000	30000	30000	0	0	0	0	6	8	0	0
959.809920	11000	27000	30000	30000	0	0	0	0	8	8	0	0
960.813284	11000	1000	30000	30000	0	0	0	0	8	1	0	0
961.816651	12000	26000	30000	30000	0	0	0	0	8	7	0	0
962.819989	11000	27000	30000	30000	0	0	0	0	7	8	0	0
963.823295	0	29000	0	0	0	0	0	0	0	8	0	0
964.826650	11000	31000	0	0	0	0	0	0	7	8	0	0
965.829916	11000	34000	0	0	0	0	0	0	8	8	0	0
966.833273	14000	1000	0	0	0	0	0	0	8	1	0	0
967.836630	18000	5000	0	0	0	0	0	0	8	5	0	0
968.839918	1000	0	0	0	0	0	0	0	1	0	0	0
969.843310	21000	0	0	0	0	0	0	0	7	0	0	0
970.846662	26000	0	0	0	0	0	0	0	8	0	0	0
971.849995	0	0	0	0	0	0	0	0	0	0	0	0
972.853311	0	0	0	0	0	0	0	0	0	0	0	0
973.856630	0	0	0	0	0	0	0	0	0	0	0	0
974.859997	0	0	0	0	0	0	0	0	0	0	0	0
975.863299	0	0	0	0	0	0	0	0	0	0	0	0
976.866646	0	0	0	0	0	0	0	0	0	0	0	0
977.869980	0	0	0	0	0	0	0	0	0	0	0	0
978.873250	0	0	0	0	0	0	0	0	0	0	0	0
979.876650	0	0	0	0	0	0	0	0	0	0	0	0
980.879978	0	0	0	0	0	0	0	0	0	0	0	0
981.883279	0	0	0	0	0	0	0	0	0	0	0	0
982.886655	0	0	0	0	0	0	0	0	0	0	0	0
983.889956	0	0	0	0	0	0	0	0	0	0	0	0
984.893294	0	0	0	0	0	0	0	0	0	0	0	0
985.896658	0	0	0	0	0	0	0	0	0	0	0	0
986.899932	0	0	0	0	0	0	0	0	0	0	0	0

A.3 Xen Limited Bandwidth Limited Client Sends Data to Server

Table A.5: Client Sending Data - Client Outgoing Bandwidth Xen Limited

	Bandwidth (bytes per second)				Bytes Dropped				Retransmitted Packets			
Time Stamp	Group 0	Group 1	Group 2	Group 3	Group 0	Group 1	Group 2	Group 3	Group 0	Group 1	Group 2	Group 3
213.640035	0	0	0	0	0	0	0	0	0	0	0	0
214.643397	0	0	0	0	0	0	0	0	0	0	0	0
215.646674	0	0	0	0	0	0	0	0	0	0	0	0
216.650036	0	0	0	0	0	0	0	0	0	0	0	0
217.653554	0	0	0	0	0	0	0	0	0	0	0	0
218.656668	0	0	0	0	0	0	0	0	0	0	0	0
219.660033	0	0	0	0	0	0	0	0	0	0	0	0
220.663371	0	0	0	0	0	0	0	0	0	0	0	0
221.666698	0	0	0	0	0	0	0	0	0	0	0	0
222.671662	0	0	0	0	0	0	0	0	0	0	0	0
223.673365	0	0	0	0	0	0	0	0	0	0	0	0
224.676694	0	0	0	0	0	0	0	0	0	0	0	0
225.680035	0	0	0	0	0	0	0	0	0	0	0	0
226.683366	0	0	0	0	0	0	0	0	0	0	0	0
227.686695	0	0	0	0	0	0	0	0	0	0	0	0
228.690034	0	0	0	0	0	0	0	0	0	0	0	0
229.693360	0	0	0	0	0	0	0	0	0	0	0	0
230.696699	0	0	0	0	0	0	0	0	0	0	0	0
231.700033	0	0	0	0	0	0	0	0	0	0	0	0
232.703362	0	0	0	0	0	0	0	0	0	0	0	0
233.706699	0	0	0	0	0	0	0	0	0	0	0	0
234.710028	0	0	0	0	0	0	0	0	0	0	0	0
235.713357	0	0	0	0	0	0	0	0	0	0	0	0
236.716694	0	0	0	0	0	0	0	0	0	0	0	0
237.720032	0	0	0	0	0	0	0	0	0	0	0	0
238.723377	0	0	0	0	0	0	0	0	0	0	0	0
239.726695	0	0	0	0	0	0	0	0	0	0	0	0
240.730016	0	0	0	0	0	0	0	0	0	0	0	0
241.733346	0	0	0	0	0	0	0	0	0	0	0	0
242.736689	0	0	0	0	0	0	0	0	0	0	0	0
243.740045	0	0	0	0	0	0	0	0	0	0	0	0
244.743334	0	0	0	0	0	0	0	0	0	0	0	0
245.746699	0	0	0	0	0	0	0	0	0	0	0	0
246.749996	0	0	0	0	0	0	0	0	0	0	0	0
247.753352	0	0	0	0	0	0	0	0	0	0	0	0
248.756693	0	0	0	0	0	0	0	0	0	0	0	0
249.760020	0	0	0	0	0	0	0	0	0	0	0	0

250.763378	0	0	0	0	0	0	0	0	0	0	0	0
251.766709	0	0	0	0	0	0	0	0	0	0	0	0
252.770007	0	0	0	0	0	0	0	0	0	0	0	0
253.773348	6000	6000	9000	7000	0	0	0	0	0	0	0	0
254.776651	1000	1000	11000	18000	0	0	0	0	1	1	7	6
255.780009	1000	1000	26000	7000	0	0	0	0	1	1	10	3
256.783372	1000	1000	11000	9000	0	0	0	0	1	1	6	5
257.786701	1000	1000	13000	22000	0	0	0	0	1	1	1	1
258.790018	0	0	11000	10000	0	0	0	0	0	0	1	0
259.793345	0	0	11000	12000	0	0	0	0	0	0	0	1
260.796659	1000	1000	11000	8000	0	0	0	0	1	1	1	0
261.800027	0	0	12000	3000	0	0	0	0	0	0	0	1
262.803334	0	0	11000	11000	0	0	0	0	0	0	1	1
263.806671	0	0	17000	17000	0	0	0	0	0	0	7	6
264.810027	0	0	15000	16000	0	0	0	0	0	0	4	7
265.813322	0	0	9000	20000	0	0	0	0	0	0	0	6
266.816654	0	0	17000	13000	0	0	0	0	0	0	7	3
267.819991	0	0	15000	16000	0	0	0	0	0	0	4	4
268.823323	1000	20000	9000	9000	0	0	0	0	1	9	0	4
269.826694	0	94000	17000	4000	0	0	0	0	0	29	7	0
270.829985	0	29000	12000	35000	0	0	0	0	0	12	1	15
271.833320	0	15000	9000	20000	0	0	0	0	0	4	0	5
272.836654	0	10000	17000	11000	0	0	0	0	0	0	7	4
273.839985	0	16000	13000	13000	0	0	0	0	0	7	3	4
274.843323	0	15000	12000	17000	0	0	0	0	0	4	2	6
275.846653	75000	11000	4000	10000	0	0	0	0	12	1	1	4
276.850029	42000	3000	30000	0	0	0	0	0	12	1	13	0
277.853362	16000	29000	15000	0	0	0	0	0	5	12	5	0
278.856695	10000	15000	11000	0	0	0	0	0	1	4	1	0
279.859991	15000	9000	17000	11000	0	0	0	0	5	0	7	2
280.863323	12000	4000	15000	67000	0	0	0	0	2	1	5	21
281.866658	15000	29000	12000	14000	0	0	0	0	5	12	2	3
282.869992	10000	15000	14000	14000	0	0	0	0	0	4	4	3
283.873342	18000	9000	15000	12000	0	0	0	0	3	0	5	2
284.876682	20000	16000	15000	5000	0	0	0	0	5	6	5	1
285.880025	23000	14000	10000	0	0	0	0	0	8	4	0	0
286.883358	15000	10000	11000	0	0	0	0	0	0	0	1	0
287.886692	16000	12000	11000	0	0	0	0	0	1	2	1	0
288.889996	20000	10000	18000	23000	0	0	0	0	5	0	8	7
289.893361	26000	11000	15000	13000	0	0	0	0	11	1	5	3
290.896693	18000	11000	10000	13000	0	0	0	0	3	1	0	3
291.900025	22000	18000	11000	13000	0	0	0	0	7	8	1	3
292.903362	17000	16000	11000	13000	0	0	0	0	2	5	1	3
293.906690	16000	9000	17000	13000	0	0	0	0	1	0	7	3
294.910004	10000	13000	15000	13000	0	0	0	0	0	2	5	3
295.913357	3000	9000	10000	9000	0	0	0	0	1	0	0	0
296.916693	5000	17000	11000	0	0	0	0	0	0	7	1	0
297.920024	5000	15000	11000	0	0	0	0	0	5	5	1	0

298.923317	61000	10000	17000	0	0	0	0	0	7	0	7	0
299.926650	25000	2000	15000	0	0	0	0	0	6	1	5	0
300.930021	1000	21000	10000	0	0	0	0	0	0	1	0	0
301.933314	38000	11000	11000	0	0	0	0	0	12	1	1	0
302.936664	27000	1000	11000	0	0	0	0	0	9	0	1	0
303.939987	1000	33000	18000	0	0	0	0	0	0	9	8	0
304.943320	40000	12000	16000	16000	0	0	0	0	13	5	6	5
305.946651	21000	5000	11000	18000	0	0	0	0	5	1	1	8
306.949986	15000	1000	17000	14000	0	0	0	0	3	1	7	3
307.953333	18000	38000	10000	4000	0	0	0	0	1	11	0	0
308.956654	15000	9000	10000	0	0	0	0	0	0	2	0	0
309.959995	5000	26000	10000	0	0	0	0	0	1	9	0	0
310.963336	0	13000	12000	0	0	0	0	0	0	4	2	0
311.966652	2000	14000	11000	0	0	0	0	0	1	4	1	0
312.969989	0	14000	17000	0	0	0	0	0	0	5	7	0
313.973328	1000	17000	14000	0	0	0	0	0	1	3	4	0
314.976648	0	11000	12000	0	0	0	0	0	0	3	2	0
315.979985	0	13000	12000	0	0	0	0	0	0	4	2	0
316.983340	0	10000	12000	0	0	0	0	0	0	0	2	0
317.986929	1000	14000	12000	0	0	0	0	0	1	1	2	0
318.990000	43000	13000	8000	0	0	0	0	0	11	3	0	0
319.993349	29000	15000	0	0	0	0	0	0	8	5	0	0
320.996667	19000	14000	21000	0	0	0	0	0	0	4	9	0
321.999980	23000	15000	13000	0	0	0	0	0	3	5	3	0
323.003317	27000	13000	13000	0	0	0	0	0	7	3	3	0
324.006667	29000	11000	13000	0	0	0	0	0	8	1	3	0
325.009987	19000	14000	13000	0	0	0	0	0	0	4	3	0
326.013318	24000	16000	13000	0	0	0	0	0	3	6	3	0
327.016644	27000	13000	13000	0	0	0	0	0	8	3	3	0
328.019985	27000	13000	9000	0	0	0	0	0	6	3	0	0
329.023325	25000	13000	0	0	0	0	0	0	6	3	0	0
330.026658	28000	13000	0	0	0	0	0	0	8	3	0	0
331.030032	0	13000	0	0	0	0	0	0	0	3	0	0
332.033353	55000	9000	0	0	0	0	0	0	15	0	0	0
333.036694	28000	0	0	0	0	0	0	0	8	0	0	0
334.040028	24000	16000	0	0	0	0	0	0	4	5	0	0
335.043341	8000	18000	0	0	0	0	0	0	2	8	0	0
336.046693	53000	13000	0	0	0	0	0	0	19	3	0	0
337.049977	24000	12000	16000	0	0	0	0	0	4	2	5	0
338.053317	29000	12000	2000	55000	0	0	0	0	9	2	0	18
339.056658	13000	4000	0	46000	0	0	0	0	3	0	0	8
340.060020	0	0	0	5000	0	0	0	0	0	0	0	1
341.063357	0	0	0	0	0	0	0	0	0	0	0	0
342.066692	0	0	0	0	0	0	0	0	0	0	0	0
343.070013	0	0	0	0	0	0	0	0	0	0	0	0
344.073325	50000	0	0	0	0	0	0	0	15	0	0	0
345.076675	33000	0	0	0	0	0	0	0	8	0	0	0
346.080022	28000	0	0	0	0	0	0	0	6	0	0	0

347.083323	0	0	0	0	0	0	0	0	0	0	0	0
348.086686	0	0	0	0	0	0	0	0	0	0	0	0
349.090018	0	0	0	0	0	0	0	0	0	0	0	0
350.093351	0	0	0	0	0	0	0	0	0	0	0	0
351.096685	0	70000	0	0	0	0	0	0	0	24	0	0
352.100018	0	16000	0	0	0	0	0	0	0	6	0	0
353.103353	0	10000	0	0	0	0	0	0	0	0	0	0
354.106681	0	12000	0	0	0	0	0	0	0	1	0	0
355.110007	0	10000	0	0	0	0	0	0	0	0	0	0
356.113348	0	15000	0	0	0	0	0	0	0	6	0	0
357.116686	1000	14000	0	0	0	0	0	0	0	4	0	0
358.120017	0	10000	0	0	0	0	0	0	0	0	0	0
359.123328	0	13000	0	0	0	0	0	0	0	2	0	0
360.126685	38000	9000	0	0	0	0	0	0	11	0	0	0
361.130024	33000	16000	0	0	0	0	0	0	8	6	0	0
362.133353	25000	16000	0	0	0	0	0	0	0	6	0	0
363.136686	33000	10000	0	0	0	0	0	0	8	0	0	0
364.140018	37000	13000	0	0	0	0	0	0	11	2	0	0
365.143324	34000	9000	0	0	0	0	0	0	10	0	0	0
366.146686	34000	15000	0	0	0	0	0	0	9	5	0	0
367.150021	25000	15000	0	0	0	0	0	0	0	5	0	0
368.153355	28000	10000	0	0	0	0	0	0	3	0	0	0
369.156644	34000	11000	10000	1000	0	0	0	0	9	1	6	0
370.160017	2000	6000	102000	0	0	0	0	0	2	1	28	0
371.163352	0	27000	2000	0	0	0	0	0	0	12	1	0
372.166686	107000	2000	1000	0	0	0	0	0	32	0	1	0
373.170017	9000	32000	0	0	0	0	0	0	0	14	0	0
374.173355	0	14000	0	0	0	0	0	0	0	4	0	0
375.176695	0	12000	0	0	0	0	0	0	0	2	0	0
376.180026	1000	13000	31000	0	0	0	0	0	1	3	12	0
377.183353	0	18000	18000	0	0	0	0	0	0	7	8	0
378.186656	68000	10000	3000	0	0	0	0	0	22	1	1	0
379.190019	36000	11000	32000	0	0	0	0	0	6	1	14	0
380.193339	9000	18000	2000	0	0	0	0	0	1	8	0	0
381.196686	77000	14000	1000	0	0	0	0	0	25	4	1	0
382.200018	38000	12000	1000	0	0	0	0	0	11	2	1	0
383.203354	0	12000	48000	0	0	0	0	0	0	2	19	0
384.206666	0	12000	17000	0	0	0	0	0	0	2	4	0
385.210005	49000	13000	0	0	0	0	0	0	16	3	0	0
386.213337	40000	11000	0	0	0	0	0	0	10	2	0	0
387.216699	24000	0	22000	0	0	0	0	0	6	0	9	0
388.220005	0	1000	49000	0	0	0	0	0	0	1	15	0
389.223333	0	0	14000	0	0	0	0	0	0	0	4	0
390.226685	0	0	14000	0	0	0	0	0	0	0	5	0
391.230014	0	0	17000	0	0	0	0	0	0	0	6	0
392.233343	0	0	9000	0	0	0	0	0	0	0	0	0
393.236684	48000	0	17000	0	0	0	0	0	16	0	7	0
394.239992	43000	0	15000	0	0	0	0	0	11	0	4	0

395.243362	27000	0	10000	0	0	0	0	0	7	0	0	0
396.246665	0	1000	15000	0	0	0	0	0	0	1	6	0
397.250012	0	0	15000	0	0	0	0	0	0	0	4	0
398.253336	0	0	9000	25000	0	0	0	0	0	0	0	10
399.256669	0	0	11000	13000	0	0	0	0	0	0	1	3
400.260013	1000	0	12000	14000	0	0	0	0	0	0	1	3
401.263339	0	0	18000	9000	0	0	0	0	0	0	8	0
402.266667	0	1000	16000	15000	0	0	0	0	0	0	7	5
403.269980	0	0	12000	14000	0	0	0	0	0	0	2	4
404.273355	0	1000	16000	12000	0	0	0	0	0	1	6	2
405.276676	0	0	14000	11000	0	0	0	0	0	0	4	2
406.280014	0	1000	15000	0	0	0	0	0	0	1	5	0
407.283346	0	0	11000	0	0	0	0	0	0	0	0	0
408.286678	0	1000	10000	0	0	0	0	0	0	1	1	0
409.289974	31000	0	16000	0	0	0	0	0	11	0	5	0
410.293308	54000	1000	2000	0	0	0	0	0	15	1	1	0
411.296650	27000	0	19000	0	0	0	0	0	2	0	1	0
412.299974	6000	20000	11000	0	0	0	0	0	1	5	1	0
413.303348	66000	5000	1000	0	0	0	0	0	18	1	0	0
414.306684	47000	1000	1000	0	0	0	0	0	17	1	1	0
415.309979	41000	1000	30000	0	0	0	0	0	11	1	1	0
416.313345	34000	1000	10000	0	0	0	0	0	9	1	0	0
417.316678	1000	0	0	0	0	0	0	0	0	0	0	0
418.319992	46000	0	12000	0	0	0	0	0	12	0	1	0
419.323347	38000	3000	15000	0	0	0	0	0	8	3	6	0
420.326640	30000	0	14000	0	0	0	0	0	0	0	4	0
421.330013	41000	0	10000	0	0	0	0	0	11	0	0	0
422.333349	44000	0	11000	0	0	0	0	0	14	0	1	0
423.336682	27000	0	11000	0	0	0	0	0	0	0	1	0
424.339989	43000	0	16000	0	0	0	0	0	11	0	6	0
425.343309	30000	0	16000	0	0	0	0	0	8	0	6	0
426.346652	31000	0	10000	0	0	0	0	0	0	0	0	0
427.349973	2000	44000	12000	0	0	0	0	0	1	16	1	0
428.353346	0	84000	5000	0	0	0	0	0	0	23	1	0
429.356639	18000	0	27000	0	0	0	0	0	3	0	12	0
430.359977	41000	0	11000	0	0	0	0	0	12	0	1	0
431.363332	36000	0	17000	1000	0	0	0	0	6	0	6	0
432.366661	12000	1000	9000	0	0	0	0	0	0	0	0	0
433.370005	56000	0	3000	0	0	0	0	0	15	0	0	0
434.373356	38000	0	0	0	0	0	0	0	8	0	0	0
435.376687	18000	0	0	0	0	0	0	0	0	0	0	0
436.380013	57000	0	5000	0	0	0	0	0	15	0	5	0
437.383348	42000	0	0	0	0	0	0	0	12	0	0	0
438.386674	13000	0	0	0	0	0	0	0	0	0	0	0
439.389976	64000	0	0	0	0	0	0	0	17	0	0	0
440.393307	46000	0	0	0	0	0	0	0	16	0	0	0
441.396678	39000	0	0	0	0	0	0	0	9	0	0	0
442.400010	37000	0	0	0	0	0	0	0	12	0	0	0

443.403310	0	11000	0	0	0	0	0	0	0	2	0	0
444.406638	2000	34000	55000	0	0	0	0	0	1	11	18	0
445.410004	0	2000	59000	0	0	0	0	0	0	1	18	0
446.413309	14000	43000	0	0	0	0	0	0	2	5	0	0
447.416645	40000	2000	0	0	0	0	0	0	11	1	0	0
448.419980	34000	0	0	0	0	0	0	0	6	0	0	0
449.423342	14000	56000	0	0	0	0	0	0	0	17	0	0
450.426687	56000	4000	0	0	0	0	0	0	15	0	0	0
451.430007	38000	0	0	0	0	0	0	0	8	0	0	0
452.433347	18000	0	0	0	0	0	0	0	0	0	0	0
453.436678	57000	1000	0	0	0	0	0	0	15	1	0	0
454.439968	47000	0	1000	0	0	0	0	0	17	0	0	0
455.443348	38000	1000	0	0	0	0	0	0	8	1	0	0
456.446675	37000	0	0	0	0	0	0	0	12	0	0	0
457.449995	1000	0	0	0	0	0	0	0	0	0	0	0
458.453340	46000	0	0	1000	0	0	0	0	12	0	0	1
459.456635	3000	80000	0	0	0	0	0	0	1	26	0	0
460.460003	1000	61000	1000	0	0	0	0	0	1	19	1	0
461.463306	42000	29000	0	0	0	0	0	0	16	9	0	0
462.466654	88000	33000	1000	1000	0	0	0	0	25	13	1	1
463.469972	40000	37000	0	0	0	0	0	0	9	13	0	0
464.473302	38000	33000	0	0	0	0	0	0	6	7	0	0
465.476664	16000	34000	0	0	0	0	0	0	0	9	0	0
466.480006	56000	35000	0	0	0	0	0	0	16	10	0	0
467.483341	36000	34000	0	0	0	0	0	0	6	9	0	0
468.486675	17000	25000	0	0	0	0	0	0	0	0	0	0
469.490009	58000	37000	0	0	0	0	0	0	15	12	0	0
470.493310	35000	40000	0	0	0	0	0	0	5	15	0	0
471.496673	12000	34000	0	0	0	0	0	0	0	9	0	0
472.500008	66000	4000	0	0	0	0	0	0	18	1	0	0
473.503321	30000	24000	0	0	0	0	0	0	6	5	0	0
474.506676	11000	61000	0	0	0	0	0	0	0	8	0	0
475.509993	2000	39000	0	0	0	0	0	0	1	13	0	0
476.513301	0	34000	0	0	0	0	0	0	0	10	0	0
477.516636	71000	2000	0	0	0	0	0	0	19	1	0	0
478.519989	26000	33000	0	0	0	0	0	0	7	2	0	0
479.523326	15000	46000	0	0	0	0	0	0	0	3	0	0
480.526647	2000	35000	0	0	0	0	0	0	1	10	0	0
481.530007	0	31000	0	0	0	0	0	0	0	6	0	0
482.533305	24000	22000	0	0	0	0	0	0	7	0	0	0
483.536632	60000	37000	0	0	0	0	0	0	18	9	0	0
484.539989	59000	33000	0	0	0	0	0	0	18	8	0	0
485.543304	46000	25000	0	0	0	0	0	0	14	0	0	0
486.546643	23000	33000	0	0	0	0	0	0	6	8	0	0
487.549969	76000	6000	0	0	0	0	0	0	24	1	0	0
488.553322	27000	29000	0	0	0	0	0	0	9	5	0	0
489.556632	56000	40000	0	0	0	0	0	0	15	1	0	0
490.559985	52000	0	0	0	0	0	0	0	15	0	0	0

491.563303	20000	0	0	0	0	0	0	0	7	0	0	0
492.566657	62000	0	0	0	0	0	0	0	17	0	0	0
493.569968	4000	22000	1000	0	0	0	0	0	1	2	1	0
494.573298	1000	36000	0	0	0	0	0	0	1	10	0	0
495.576637	0	35000	0	0	0	0	0	0	0	6	0	0
496.580014	6000	15000	0	0	0	0	0	0	4	3	0	0
497.583342	0	0	0	0	0	0	0	0	0	0	0	0
498.586691	0	0	0	0	0	0	0	0	0	0	0	0
499.590005	0	0	0	0	0	0	0	0	0	0	0	0
500.593339	86000	0	0	0	0	0	0	0	29	0	0	0
501.596681	31000	0	0	0	0	0	0	0	6	0	0	0
502.600011	0	61000	0	0	0	0	0	0	0	21	0	0
503.603333	0	38000	0	0	0	0	0	0	0	8	0	0
504.606668	0	16000	0	0	0	0	0	0	0	4	0	0
505.610036	0	0	0	0	0	0	0	0	0	0	0	0
506.613342	0	0	0	0	0	0	0	0	0	0	0	0
507.616668	0	0	0	0	0	0	0	0	0	0	0	0
508.620001	50000	0	0	0	0	0	0	0	15	0	0	0
509.623339	36000	0	0	0	0	0	0	0	6	0	0	0
510.626675	25000	0	0	0	0	0	0	0	4	0	0	0
511.629997	0	0	0	0	0	0	0	0	0	0	0	0
512.633365	0	0	0	0	0	0	0	0	0	0	0	0
513.636682	0	0	0	0	0	0	0	0	0	0	0	0
514.640006	0	0	0	0	0	0	0	0	0	0	0	0
515.643330	0	0	0	0	0	0	0	0	0	0	0	0
516.646701	0	0	0	0	0	0	0	0	0	0	0	0
517.650045	0	0	0	0	0	0	0	0	0	0	0	0
518.653337	0	70000	0	11000	0	0	0	0	0	22	0	1
519.656630	0	40000	0	17000	0	0	0	0	0	10	0	7
520.660553	0	7000	27000	17000	0	0	0	0	0	2	9	7
521.663338	0	0	12000	5000	0	0	0	0	0	0	2	1
522.666674	0	0	12000	23000	0	0	0	0	0	0	2	7
523.669967	0	0	19000	12000	0	0	0	0	0	0	4	6
524.673337	62000	0	19000	11000	0	0	0	0	22	0	4	3
525.676664	40000	1000	19000	0	0	0	0	0	11	0	4	0
526.680003	38000	0	3000	44000	0	0	0	0	10	0	0	18
527.683333	0	0	0	12000	0	0	0	0	0	0	0	2
528.686667	0	0	0	17000	0	0	0	0	0	0	0	7
529.689980	0	0	0	15000	0	0	0	0	0	0	0	4
530.693335	0	0	0	9000	0	0	0	0	0	0	0	0
531.696671	0	0	0	17000	0	0	0	0	0	0	0	7
532.700001	0	0	0	15000	0	0	0	0	0	0	0	4
533.703334	0	0	0	9000	0	0	0	0	0	0	0	0
534.706669	0	0	0	17000	0	0	0	0	0	0	0	7
535.709972	0	0	0	12000	0	0	0	0	0	0	0	1
536.713329	0	0	0	9000	0	0	0	0	0	0	0	0
537.716668	0	0	0	17000	0	0	0	0	0	0	0	7
538.720002	0	0	0	12000	0	0	0	0	0	0	0	1

539.723335	0	0	0	9000	0	0	0	0	0	0	0	0
540.726655	0	0	0	17000	0	0	0	0	0	0	0	7
541.729981	0	0	0	15000	0	0	0	0	0	0	0	4
542.733329	0	0	0	9000	0	0	0	0	0	0	0	0
543.736668	0	0	0	16000	0	0	0	0	0	0	0	6
544.739998	0	0	0	14000	0	0	0	0	0	0	0	4
545.743334	0	0	0	10000	0	0	0	0	0	0	0	0
546.746670	0	0	0	11000	0	0	0	0	0	0	0	1
547.749992	0	0	0	11000	0	0	0	0	0	0	0	1
548.753334	0	0	0	18000	0	0	0	0	0	0	0	8
549.756625	0	0	0	16000	0	0	0	0	0	0	0	6
550.760000	0	77000	0	5000	0	0	0	0	0	23	0	0
551.763314	0	35000	0	28000	0	0	0	0	0	10	0	13
552.766666	0	0	0	12000	0	0	0	0	0	0	0	2
553.769996	0	0	0	17000	0	0	0	0	0	0	0	7
554.773290	0	0	0	16000	0	0	0	0	0	0	0	6
555.776656	1000	0	1000	10000	0	0	0	0	0	0	0	0
556.780139	42000	1000	0	2000	0	0	0	0	11	0	0	1
557.783297	38000	0	0	1000	0	0	0	0	10	0	0	1
558.786649	30000	0	0	1000	0	0	0	0	8	0	0	1
559.789966	0	0	0	24000	0	0	0	0	0	0	0	1
560.793331	0	0	0	15000	0	0	0	0	0	0	0	0
561.796667	0	0	0	1000	0	0	0	0	0	0	0	0
562.799997	0	0	0	11000	0	0	0	0	0	0	0	1
563.803296	0	0	0	16000	0	0	0	0	0	0	0	6
564.806649	0	0	0	15000	0	0	0	0	0	0	0	5
565.809964	0	0	0	10000	0	0	0	0	0	0	0	0
566.813334	0	0	0	11000	0	0	0	0	0	0	0	1
567.816667	0	0	0	11000	0	0	0	0	0	0	0	1
568.819969	0	0	0	16000	0	0	0	0	0	0	0	6
569.823319	0	0	0	14000	0	0	0	0	0	0	0	4
570.826662	0	0	0	10000	0	0	0	0	0	0	0	0
571.829981	0	0	0	12000	0	0	0	0	0	0	0	2
572.833336	0	0	0	10000	0	0	0	0	0	0	0	0
573.836666	0	0	0	12000	0	0	0	0	0	0	0	1
574.839990	0	0	0	10000	0	0	0	0	0	0	0	1
575.843302	0	0	0	11000	0	0	0	0	0	0	0	1
576.846624	0	0	0	18000	0	0	0	0	0	0	0	8
577.849960	0	0	0	16000	0	0	0	0	0	0	0	6
578.853331	0	0	0	10000	0	0	0	0	0	0	0	0
579.856666	0	0	0	11000	0	0	0	0	0	0	0	1
580.859981	0	0	21000	11000	0	0	0	0	0	0	5	1
581.863296	0	0	27000	18000	0	0	0	0	0	0	9	8
582.866627	0	0	31000	16000	0	0	0	0	0	0	7	6
583.869959	0	0	31000	16000	0	0	0	0	0	0	8	6
584.873333	0	0	7000	10000	0	0	0	0	0	0	2	0
585.876657	0	0	0	11000	0	0	0	0	0	0	0	1
586.879995	1000	0	1000	11000	0	0	0	0	0	0	0	1

587.883328	0	0	0	20000	0	0	0	0	0	0	0	10
588.886642	0	0	0	10000	0	0	0	0	0	0	0	0
589.889993	0	0	0	11000	0	0	0	0	0	0	0	1
590.893333	0	0	0	11000	0	0	0	0	0	0	0	1
591.896664	0	0	0	19000	0	0	0	0	0	0	0	9
592.899997	0	0	0	10000	0	0	0	0	0	0	0	0
593.903327	0	0	0	11000	0	0	0	0	0	0	0	1
594.906661	0	0	0	11000	0	0	0	0	0	0	0	1
595.909996	0	0	0	17000	0	0	0	0	0	0	0	7
596.913298	0	0	0	15000	0	0	0	0	0	0	0	5
597.916631	0	0	0	12000	0	0	0	0	0	0	0	2
598.919994	0	0	0	16000	0	0	0	0	0	0	0	6
599.923297	0	0	0	17000	0	0	0	0	0	0	0	7
600.926664	0	0	0	12000	0	0	0	0	0	0	0	2
601.929997	0	0	0	16000	0	0	0	0	0	0	0	6
602.933327	0	0	0	14000	0	0	0	0	0	0	0	4
603.936665	0	0	0	15000	0	0	0	0	0	0	0	5
604.939991	0	0	0	10000	0	0	0	0	0	0	0	0
605.943310	0	0	0	11000	0	0	0	0	0	0	0	1
606.946663	0	0	0	11000	0	0	0	0	0	0	0	1
607.949994	0	0	0	18000	0	0	0	0	0	0	0	8
608.953327	0	0	0	16000	0	0	0	0	0	0	0	6
609.956668	0	0	0	10000	0	0	0	0	0	0	0	0
610.959999	0	45000	0	11000	0	0	0	0	0	11	0	1
611.963332	0	40000	0	11000	0	0	0	0	0	10	0	1
612.966663	0	29000	0	16000	0	0	0	0	0	8	0	6
613.969994	0	0	0	16000	0	0	0	0	0	0	0	6
614.973299	0	0	0	10000	0	0	0	0	0	0	0	0
615.976667	0	0	0	11000	0	0	0	0	0	0	0	1
616.979956	32000	0	0	11000	0	0	0	0	9	0	0	1
617.983293	41000	0	0	16000	0	0	0	0	11	0	0	6
618.986639	38000	1000	0	15000	0	0	0	0	9	0	0	5
619.989994	16000	0	0	10000	0	0	0	0	3	0	0	0
620.993333	0	0	0	11000	0	0	0	0	0	0	0	1
621.996665	0	0	0	11000	0	0	0	0	0	0	0	1
622.999951	0	0	0	17000	0	0	0	0	0	0	0	7
624.003330	0	0	0	15000	0	0	0	0	0	0	0	5
625.006660	0	0	0	10000	0	0	0	0	0	0	0	0
626.010000	0	0	0	11000	0	0	0	0	0	0	0	1
627.013304	0	0	0	11000	0	0	0	0	0	0	0	1
628.016642	0	0	0	17000	0	0	0	0	0	0	0	7
629.020127	0	0	0	17000	0	0	0	0	0	0	0	7
630.023296	0	0	0	12000	0	0	0	0	0	0	0	2
631.026665	0	0	0	18000	0	0	0	0	0	0	0	8
632.029992	0	0	0	13000	0	0	0	0	0	0	0	3
633.033323	0	0	0	14000	0	0	0	0	0	0	0	4
634.036661	0	0	0	10000	0	0	0	0	0	0	0	0
635.040002	0	0	0	11000	0	0	0	0	0	0	0	1

636.043298	0	0	0	11000	0	0	0	0	0	0	0	1
637.046659	0	0	0	17000	0	0	0	0	0	0	0	7
638.049993	0	0	0	15000	0	0	0	0	0	0	0	5
639.053324	0	0	0	10000	0	0	0	0	0	0	0	0
640.056616	0	0	0	11000	0	0	0	0	0	0	0	1
641.059975	0	0	0	15000	0	0	0	0	0	0	0	5
642.063295	0	0	64000	12000	0	0	0	0	0	0	20	1
643.066661	0	0	37000	17000	0	0	0	0	0	0	7	3
644.069992	0	0	12000	23000	0	0	0	0	0	0	2	8
645.073325	0	0	0	26000	0	0	0	0	0	0	0	11
646.076651	0	0	0	21000	0	0	0	0	0	0	0	6
647.079987	0	0	0	21000	0	0	0	0	0	0	0	6
648.083325	0	0	1000	15000	0	0	0	0	0	0	0	0
649.086655	0	0	0	16000	0	0	0	0	0	0	0	1
650.089960	1000	0	0	20000	0	0	0	0	0	0	0	5
651.093322	0	0	0	26000	0	0	0	0	0	0	0	11
652.096658	0	0	0	21000	0	0	0	0	0	0	0	6
653.099985	0	0	0	19000	0	0	0	0	0	0	0	4
654.103322	0	0	0	15000	0	0	0	0	0	0	0	0
655.106660	0	0	0	16000	0	0	0	0	0	0	0	1
656.109990	0	0	0	22000	0	0	0	0	0	0	0	7
657.113324	0	0	0	21000	0	0	0	0	0	0	0	6
658.116657	0	0	0	15000	0	0	0	0	0	0	0	0
659.119991	0	0	0	17000	0	0	0	0	0	0	0	1
660.123324	0	0	0	17000	0	0	0	0	0	0	0	3
661.126700	0	0	0	23000	0	0	0	0	0	0	0	8
662.129992	0	0	0	20000	0	0	0	0	0	0	0	5
663.133321	0	0	0	22000	0	0	0	0	0	0	0	7
664.136655	0	0	0	15000	0	0	0	0	0	0	0	0
665.139992	0	0	0	17000	0	0	0	0	0	0	0	1
666.143320	0	0	0	18000	0	0	0	0	0	0	0	4
667.146652	0	0	0	21000	0	0	0	0	0	0	0	6
668.149989	0	0	0	23000	0	0	0	0	0	0	0	8
669.153319	0	0	0	15000	0	0	0	0	0	0	0	0
670.156618	0	0	0	17000	0	0	0	0	0	0	0	1
671.159988	0	55000	0	3000	0	0	0	0	0	17	0	0
672.163323	0	45000	0	3000	0	0	0	0	0	15	0	3
673.166618	0	43000	0	0	0	0	0	0	0	13	0	0
674.169951	0	27000	0	90000	0	0	0	0	0	10	0	24
675.173322	0	60000	0	9000	0	0	0	0	0	17	0	4
676.176655	0	44000	0	1000	0	0	0	0	0	14	0	1
677.180530	6000	40000	0	1000	0	0	0	0	4	10	0	1
678.183282	42000	28000	0	25000	0	0	0	0	11	0	0	9
679.186617	2000	1000	0	97000	0	0	0	0	0	1	0	30
680.189999	0	0	0	46000	0	0	0	0	0	0	0	14
681.193314	0	42000	0	26000	0	0	0	0	0	10	0	6
682.196702	0	33000	0	20000	0	0	0	0	0	8	0	0
683.199991	0	26000	0	24000	0	0	0	0	0	0	0	4

684.203283	0	32000	0	29000	0	0	0	0	0	9	0	8
685.206656	0	17000	0	2000	0	0	0	0	0	0	0	1
686.209969	0	61000	0	1000	0	0	0	0	0	19	0	1
687.213348	0	36000	0	1000	0	0	0	0	0	9	0	1
688.216657	0	53000	0	0	0	0	0	0	0	16	0	0
689.219943	0	34000	0	5000	0	0	0	0	0	9	0	5
690.223322	0	32000	0	0	0	0	0	0	0	3	0	0
691.226613	0	14000	0	108000	0	0	0	0	0	1	0	30
692.229966	0	49000	0	34000	0	0	0	0	0	13	0	14
693.233306	0	36000	0	30000	0	0	0	0	0	6	0	10
694.236645	0	19000	0	32000	0	0	0	0	0	0	0	11
695.239978	0	57000	0	5000	0	0	0	0	0	16	0	0
696.243310	0	47000	0	1000	0	0	0	0	0	17	0	1
697.246664	0	4000	0	78000	0	0	0	0	0	0	0	24
698.250013	0	67000	0	27000	0	0	0	0	0	23	0	7
699.253306	0	60000	0	2000	0	0	0	0	0	17	0	0
700.256652	0	39000	0	1000	0	0	0	0	0	10	0	1
701.259967	0	38000	0	1000	0	0	0	0	0	8	0	1
702.263322	0	20000	72000	1000	0	0	0	0	0	4	23	1
703.266641	0	7000	40000	1000	0	0	0	0	0	3	10	0
704.269970	0	0	45000	1000	0	0	0	0	0	0	15	1
705.273303	0	1000	42000	0	0	0	0	0	0	1	12	0
706.276638	0	0	29000	0	0	0	0	0	0	0	0	0
707.279981	0	3000	43000	0	0	0	0	0	0	3	12	0
708.283293	0	0	27000	60000	0	0	0	0	0	0	11	18
709.286610	0	61000	26000	37000	0	0	0	0	0	21	9	12
710.289991	0	2000	48000	35000	0	0	0	0	0	0	18	10
711.293495	0	0	84000	1000	0	0	0	0	0	0	27	0
712.296623	0	0	46000	0	0	0	0	0	0	0	16	0
713.299983	0	0	38000	0	0	0	0	0	0	0	8	0
714.303295	0	0	28000	0	0	0	0	0	0	0	0	0
715.306632	0	0	26000	0	0	0	0	0	0	0	6	0
716.309941	0	0	20000	63000	0	0	0	0	0	0	6	18
717.313279	0	63000	6000	8000	0	0	0	0	0	20	5	2
718.316612	0	47000	56000	0	0	0	0	0	0	10	1	0
719.319941	0	50000	43000	0	0	0	0	0	0	14	11	0
720.323277	0	29000	36000	0	0	0	0	0	0	10	6	0
721.326608	0	52000	21000	0	0	0	0	0	0	13	0	0
722.329945	0	35000	53000	0	0	0	0	0	0	6	14	0
723.333275	0	50000	38000	0	0	0	0	0	0	15	8	0
724.336613	0	37000	18000	0	0	0	0	0	0	9	0	0
725.339957	0	30000	59000	0	0	0	0	0	0	8	17	0
726.343315	0	0	36000	0	0	0	0	0	0	0	6	0
727.346625	0	28000	16000	0	0	0	0	0	0	9	0	0
728.349984	0	35000	60000	0	0	0	0	0	0	12	16	0
729.353278	0	48000	47000	0	0	0	0	0	0	18	17	0
730.356608	0	57000	38000	0	0	0	0	0	0	20	8	0
731.359946	0	47000	37000	0	0	0	0	0	0	13	12	0

732.363286	0	32000	0	35000	0	0	0	0	0	11	0	14
733.366612	0	2000	10000	84000	0	0	0	0	0	2	1	23
734.369944	0	0	12000	0	0	0	0	0	0	0	5	0
735.373278	0	1000	77000	0	0	0	0	0	0	1	21	0
736.376638	0	0	19000	0	0	0	0	0	0	0	6	0
737.379985	0	0	1000	0	0	0	0	0	0	0	0	0
738.383320	1000	0	51000	0	0	0	0	0	1	0	12	0
739.386629	0	117000	2000	0	0	0	0	0	0	36	1	0
740.389940	0	49000	53000	0	0	0	0	0	0	17	3	0
741.393278	0	53000	51000	0	0	0	0	0	0	20	12	0
742.396650	0	3000	36000	1000	0	0	0	0	0	0	6	0
743.399973	1000	23000	21000	0	0	0	0	0	1	10	0	0
744.403286	0	94000	2000	0	0	0	0	0	0	26	1	0
745.406656	0	50000	1000	0	0	0	0	0	0	14	1	0
746.409981	0	36000	51000	0	0	0	0	0	0	6	13	0
747.413314	0	20000	38000	0	0	0	0	0	0	0	8	0
748.416649	0	54000	1000	0	0	0	0	0	0	14	0	0
749.419939	0	38000	52000	0	0	0	0	0	0	8	15	0
750.423300	0	18000	58000	0	0	0	0	0	0	0	15	0
751.426648	0	57000	1000	0	0	0	0	0	0	15	0	0
752.429943	0	37000	12000	0	0	0	0	0	0	7	2	0
753.433383	0	19000	61000	0	0	0	0	0	0	0	16	0
754.436634	0	55000	20000	0	0	0	0	0	0	14	6	0
755.439939	0	35000	16000	0	0	0	0	0	0	8	4	0
756.443315	0	21000	68000	0	0	0	0	0	0	0	21	0
757.446610	0	30000	40000	0	0	0	0	0	0	6	10	0
758.449955	0	43000	1000	0	0	0	0	0	0	12	0	0
759.453272	0	32000	84000	0	0	0	0	0	0	4	25	0
760.456648	0	8000	33000	0	0	0	0	0	0	0	6	0
761.459951	0	57000	1000	0	0	0	0	0	0	16	0	0
762.463314	0	36000	43000	0	0	0	0	0	0	6	11	0
763.466629	0	17000	47000	0	0	0	0	0	0	0	17	0
764.469940	0	60000	4000	11000	0	0	0	0	0	17	0	3
765.473275	0	3000	32000	62000	0	0	0	0	0	1	13	20
766.476601	0	1000	92000	5000	0	0	0	0	0	1	25	1
767.479949	0	88000	9000	5000	0	0	0	0	0	28	2	5
768.483317	0	81000	3000	1000	0	0	0	0	0	23	3	1
769.486642	0	36000	0	0	0	0	0	0	0	6	0	0
770.489936	0	19000	11000	5000	0	0	0	0	0	0	5	5
771.493281	0	1000	102000	0	0	0	0	0	0	1	32	0
772.496646	0	0	0	117000	0	0	0	0	0	0	0	35
773.499936	0	44000	1000	50000	0	0	0	0	0	12	0	16
774.503270	0	37000	1000	44000	0	0	0	0	0	11	1	14
775.506625	0	31000	51000	5000	0	0	0	0	0	0	15	1
776.509939	0	3000	40000	51000	0	0	0	0	0	1	10	5
777.513283	0	56000	23000	45000	0	0	0	0	0	16	9	10
778.516646	0	49000	1000	49000	0	0	0	0	0	19	0	14
779.519975	0	39000	61000	1000	0	0	0	0	0	8	16	0

780.523272	0	45000	2000	27000	0	0	0	0	0	16	1	10
781.526604	0	25000	40000	64000	0	0	0	0	0	13	3	18
782.529936	0	54000	4000	1000	0	0	0	0	0	2	1	1
783.533277	0	18000	0	0	0	0	0	0	0	4	0	0
784.536602	0	0	43000	85000	0	0	0	0	0	0	12	29
785.539976	0	1000	66000	4000	0	0	0	0	0	1	18	1
786.543328	0	0	2000	5000	0	0	0	0	0	0	0	5
787.546610	0	59000	27000	1000	0	0	0	0	0	17	7	1
788.549964	0	3000	64000	5000	0	0	0	0	0	1	17	1
789.553317	0	1000	34000	74000	0	0	0	0	0	1	8	21
790.556605	0	113000	1000	3000	0	0	0	0	0	34	0	0
791.559970	0	5000	76000	1000	0	0	0	0	0	3	22	1
792.563313	0	0	40000	5000	0	0	0	0	0	0	10	5
793.566644	0	53000	21000	1000	0	0	0	0	0	16	6	1
794.569975	0	35000	0	0	0	0	0	0	0	5	0	0
795.573270	0	12000	0	47000	0	0	0	0	0	0	0	13
796.576645	0	67000	0	37000	0	0	0	0	0	19	0	10
797.579935	0	45000	5000	0	0	0	0	0	0	16	4	0
798.583286	115000	10000	0	0	0	0	0	0	35	6	0	0
799.586645	4000	73000	0	1000	0	0	0	0	2	16	0	1
800.589946	0	46000	0	0	0	0	0	0	0	16	0	0
801.593307	0	38000	0	14000	0	0	0	0	0	8	0	8
802.596602	0	1000	0	76000	0	0	0	0	0	0	0	16
803.599933	0	68000	0	32000	0	0	0	0	0	16	0	9
804.603269	0	40000	0	0	0	0	0	0	0	12	0	0
805.606614	1000	18000	1000	54000	0	0	0	0	0	4	1	20
806.609934	0	0	0	94000	0	0	0	0	0	0	0	25
807.613305	0	1000	0	23000	0	0	0	0	0	1	0	8
808.616599	0	0	0	63000	0	0	0	0	0	0	0	20
809.619940	0	1000	0	47000	0	0	0	0	0	1	0	12
810.623306	0	0	0	20000	0	0	0	0	0	0	0	8
811.626605	0	0	0	63000	0	0	0	0	0	0	0	19
812.629937	0	0	0	38000	0	0	0	0	0	0	0	10
813.633312	0	1000	0	28000	0	0	0	0	0	1	0	10
814.636655	0	0	0	0	0	0	0	0	0	0	0	0
815.639977	0	0	0	0	0	0	0	0	0	0	0	0
816.643313	0	0	0	0	0	0	0	0	0	0	0	0
817.646632	0	0	0	0	0	0	0	0	0	0	0	0
818.649971	0	0	0	0	0	0	0	0	0	0	0	0
819.653299	0	5000	0	0	0	0	0	0	0	5	0	0
820.656630	0	0	0	0	0	0	0	0	0	0	0	0
821.659974	0	37000	1000	0	0	0	0	0	0	12	1	0
822.663310	0	1000	104000	0	0	0	0	0	0	1	30	0
823.666651	0	0	0	0	0	0	0	0	0	0	0	0
824.669984	0	4000	0	0	0	0	0	0	0	3	0	0
825.673264	0	0	0	0	0	0	0	0	0	0	0	0
826.676636	0	0	0	0	0	0	0	0	0	0	0	0
827.679978	0	0	0	0	0	0	0	0	0	0	0	0

828.683328	0	0	0	0	0	0	0	0	0	0	0	0
829.686663	0	0	0	0	0	0	0	0	0	0	0	0
830.689953	0	0	0	0	0	0	0	0	0	0	0	0
831.693331	0	0	0	0	0	0	0	0	0	0	0	0
832.696644	0	0	0	0	0	0	0	0	0	0	0	0
833.700001	0	0	0	0	0	0	0	0	0	0	0	0
834.703309	0	0	0	0	0	0	0	0	0	0	0	0
835.706660	0	0	1000	0	0	0	0	0	0	0	0	0
836.709970	0	0	0	0	0	0	0	0	0	0	0	0
837.713333	0	0	0	0	0	0	0	0	0	0	0	0
838.716650	0	0	0	0	0	0	0	0	0	0	0	0
839.720001	0	0	0	0	0	0	0	0	0	0	0	0
840.723292	0	0	0	0	0	0	0	0	0	0	0	0
841.726611	0	0	0	0	0	0	0	0	0	0	0	0
842.729933	0	0	0	0	0	0	0	0	0	0	0	0
843.733302	0	0	0	0	0	0	0	0	0	0	0	0
844.736644	0	0	0	0	0	0	0	0	0	0	0	0
845.739975	0	0	0	0	0	0	0	0	0	0	0	0
846.743310	0	0	0	0	0	0	0	0	0	0	0	0
847.746634	0	0	0	0	0	0	0	0	0	0	0	0
848.749969	0	0	0	0	0	0	0	0	0	0	0	0
849.753269	0	0	0	0	0	0	0	0	0	0	0	0
850.756634	0	0	0	0	0	0	0	0	0	0	0	0
851.759965	0	0	0	0	0	0	0	0	0	0	0	0
852.763308	0	0	0	0	0	0	0	0	0	0	0	0
853.766640	0	0	0	0	0	0	0	0	0	0	0	0
854.769969	0	0	3000	0	0	0	0	0	0	0	3	0

Table A.6: Xen Bridge - Open vSwitch

	Xen Bridge				Open vSwitch			
MBits/sec	**CPU %**	**VM1 to VM2**	**VM2 to PM**	**PM to VM1**	**CPU %**	**VM1 to VM2**	**VM2 to PM**	**PM to VM1**
5	22.3	4.98	5.05	4.98	24.1	4.98	5.04	4.98
	22.8	4.98	4.98	5.04	22.1	4.98	4.98	4.98
	22	5.05	4.98	4.98	23	5.05	4.98	5.05
	21.2	4.98	5.05	4.98	22.8	4.98	4.98	4.98
	23.6	4.98	4.98	4.98	23.1	4.98	5.05	4.98
	21.8	5.04	4.98	5.04	23.4	5.05	4.98	4.98
	20.9	4.98	5.05	4.98	20.7	4.98	4.98	5.05
	22.2	4.98	4.98	4.98	21.5	4.98	5.05	4.98
	21.4	5.05	5.04	5.05	22.6	4.98	4.98	4.98
	22	4.98	4.98	4.98	24.2	5.05	4.98	4.98
10	35.9	10	10	9.9	38.5	9.96	10	10
	40.2	10	9.92	10.1	37.6	9.96	9.96	10
	38.8	10	10.1	9.95	42.7	10.1	10	9.96
	38.1	9.96	9.96	10	38.3	9.96	9.83	10
	37.4	10	10	9.95	41.9	10	10.2	9.97
	37.8	9.96	10	10	39.8	9.96	9.96	10
	37.7	10	9.96	9.95	38.6	10	10	10
	37	9.96	10.1	10	37.6	9.96	10	9.95
	37.9	10	9.97	10	36.2	10	9.96	10
	38.3	10	10	9.97	37.9	10	10	9.95
15	52.5	14.9	15.1	14.9	54.5	15.1	14.7	14.9
	53.7	15	14.9	15	54.3	15.1	15.3	15
	53.7	15	15.1	15	54.9	14.6	14.8	15
	51.9	14.9	14.9	15	54.4	15.1	15.2	15
	52.7	15.1	15	15	53.7	15.1	15.1	15
	53.2	15	15	15	54	15	14.9	15
	51.8	15.1	14.9	15	54.4	15	15	15
	52.6	14.9	15.1	15	53.7	14.9	14.9	15
	52.2	14.9	15	15	53.1	15.1	14.9	15
	54.8	15.1	14.9	15	52.1	14.9	15.1	15
20	67.7	20.1	20.1	20	68.8	19.9	20	20
	68.2	19.9	20	20	68.9	19.9	20	20
	72.5	20.1	20	20	70.9	20.1	19.8	20
	69.5	20	20	20	68.1	19.9	20.5	20
	67.4	19.9	20	20	66.8	19.9	20	20.1
	65.5	19.9	20	20	70.2	20.1	19.9	19.9

	68.5	19.9	20.1	20	64.8	20.1	19.7	20.1
	67.8	20.2	20	20	65.7	20	20.4	20
	64.2	19.9	20	20	66	19.9	20.1	20
	65.7	20.1	19.9	20.1	67.3	20.2	19.8	20
25	78.7	25.7	26.9	25	87.3	25.6	25.2	25
	84.4	24.7	22.1	25.1	86.5	25.4	27.3	25
	81.4	25	26.9	25	87.2	24.8	23.9	25
	81.7	25.2	24	25	86.7	25.4	23.7	25
	83	24.4	23.9	25	83.7	24.8	26	25
	84.6	24.1	26.9	25	88.2	24.6	24.2	25
	85.7	25	25.4	25	87.9	24.5	25.7	25
	79.6	25	25	24.9	86.5	24.6	24.5	25
	82.5	24.5	24.7	25	87.4	25.4	26.6	25
	82.4	26.1	24.8	25	84.5	26.1	25	25
30	98.7	29.8	28.7	30	104	28.1	32.7	30
	100.9	28	29.9	30	104.3	31.9	28	30
	100.3	29.9	30.9	30	101.9	28	30.9	30
	100	31.1	31.1	30	101.7	30.7	29.1	29.9
	102.2	28.8	30.1	29.9	103.4	29	32.3	30
	99.9	29.8	28.5	30	101.1	31.3	28.2	30
	102.3	32.7	31.1	30	103.8	29.3	29.5	30
	101.4	28.8	28.9	30	100	29	30.6	29.9
	101.8	31.1	28.8	30	103.9	30.7	30.1	30.1
	96.7	27.6	30.9	30	101.8	30.9	30.1	29.9
35	118.3	34.8	33.7	35	113	33.3	30.7	35
	119.7	36.5	37.2	35	111.9	37.2	40.2	35
	117	36.5	33.5	35.1	112.1	32.6	34.6	35
	120	32.4	36.6	35	115.6	37	34.5	35
	121.7	35.5	34.3	35	113.7	34.8	36.3	35
	119.1	37	35.2	35	112.2	35.2	35.1	35
	119.2	33.6	32.3	35	111.9	35.5	32.8	35
	115.7	34.4	38.5	35	114.8	33.6	34.7	35
	118.9	34	34.1	35	111.5	35.9	34.4	35
	115.4	35.6	37.7	35	113.2	33.5	32.7	35
40	124.6	39.7	26.8	40	122.1	40.1	23.6	40
	125.6	39.2	30.6	40	124.1	38.6	38.2	40
	123.5	41.3	31.3	40	125	41.3	31.1	40
	122.7	39.6	38.9	40	122	40.1	35.5	40
	123.6	39.7	33.1	40	124	39	32.1	40
	127.5	40.6	34.3	40	124.3	41.2	34	40
	125.1	40.6	30	40	124.4	38	34.1	40
	122.3	40.6	31.9	40	130.9	41.8	32.8	40
	120	38.4	33.2	40	117.8	39.8	35.2	40
	118.1	39.6	30.6	40	124.8	39.2	35.1	39.9
45	123	44.9	25.8	45	129.4	47.4	20.1	45
	120.1	44.8	25.8	45	124.6	44.2	23.7	45

	115.9	45.9	24	45	128.6	45.8	21.9	45
	124.3	47.4	18.9	45	126.7	45.2	23.3	45
	120.8	42.6	22.7	45	125.5	44.9	21.1	45
	120.1	45.1	18.6	45	123.3	43	20.4	45
	127.1	43.9	19.7	45	125.6	47.2	25	45
	125.4	45	21.7	44.9	121.3	43.5	16.6	45
	117.7	46.8	22.6	45	126.4	47.1	23	45
	118.7	43.8	24.4	45	126.6	44.7	28	45.1
50	127.6	42.5	14.7	50.1	125.1	46.8	15.4	50
	131.7	49.2	20.4	49.9	128.7	49.1	13.5	50
	129.6	45.5	18.7	50.1	129.9	48.5	17.4	49.9
	129.8	47.1	19.5	49.9	128	48.6	14.9	50.1
	125.7	47.9	19.7	50.1	128.2	46.3	17.8	49.9
	130.3	47	18.5	49.9	122.3	49	13.7	50.1
	118.6	46.5	13.6	50	124.3	45.9	15.9	49.9
	127.6	45.7	22.3	50	120.8	47.9	17.8	50
	124.3	45.6	17.6	49.9	121.5	48.2	16.5	50.1
	123.7	45	21.5	50	128.7	47.5	12.6	50
55	127.7	43.6	13.2	54.9	122.8	47.2	19	55
	123.4	49.7	10.7	55	125.2	48.5	13.1	55
	130.3	44.8	17.9	54.9	122.5	49.6	14.8	55.1
	125.7	47.4	18.2	55.1	122.4	47.4	20.5	55
	123.2	46.9	19.3	55	127.1	45.9	15	55
	127.3	45.1	18.1	55	129.1	47.7	18.5	55
	124.7	47.6	17.3	55	130.3	47.4	12.5	55.1
	123.1	47.7	23.7	55	131.9	49.1	17.5	54.9
	128.5	48.1	19.6	55	125.5	42.9	13.4	54.9
	128.2	46	33.3	55.1	125.5	47.7	13.7	54.9

This source listing contains the primary (and relevant) logic for the bandwidth controller in the proposed cloud environment. There are other smaller changes that were made as part of this work which are hosted on the GitHub repository hosted by Texas State University Department of Computer Science. These smaller changes were made to the SCTP kernel module for Linux and are for interfacing purposes. This code is designed to be generic to the transport algorithm, which is why the interface code is not shown here. Please contact the author for access to the total source package as hosted on the Texas State CS GitHub.

```
#include <linux/spinlock.h>
#include <linux/spinlock_types.h>
#include <linux/workqueue.h>
#include <linux/sched.h>
#include <linux/list.h>
#include <linux/slab.h>
#include <net/sctp/sctp.h>
#include <net/sctp/sm.h>
#include <net/sctp/structs.h>

// fixed point multiplier
#define FP_MULT 1000
#define DATA_CHUNK_HDR_LEN      16

// we define the stream-association definition
struct StreamAssoc
{
        struct list_head list;
        struct sctp_association* assocPtr;
};

// for now the groups are static. each group is really a
// structure with management metrics and a queue of
// stream associations
struct StreamAssocGroup
{
        // linked list of endpointStreams
        struct list_head streamAssocList;

        // number of bytes sent during a timeslice
        u64 byteCount;

        // portion of bandwidth dedicated to this group
        u32 bandwidthDivisor;

        // this is provided by the user
        u64 configuredByteLimit;

        // this is calculated on the timer
        u64 actualByteLimit;

        // number of retransmissions incurred, reset on timer
        u64 rtxChunkCount;

        // number of bytes dropped since last quanta
        u64 bytesDropped;
};

// create an array of partitions
struct StreamAssocGroup streamAssocSendGroups[MaxBandwidthGroups];
struct StreamAssocGroup streamAssocRecvGroups[MaxBandwidthGroups];

// maximum bandwidth limit, bytes * FP_MULT. set by the user
u64 MaxBandwidthSend = 0xffffffffffffffff;
u64 MaxBandwidthRecv = 0xffffffffffffffff;

// the maximum bandwidth requested, in bytes.
u64 ReqMaxBandwidthSend = 0;
u64 ReqMaxBandwidthRecv = 0;

// amount of available bandwidth to use in adjusting limits
u64 SendBandwidthFreePool = 0;
u64 RecvBandwidthFreePool = 0;

// bandwidth counts for used across all groups
u64 SendBandwidthUsed = 0;
u64 RecvBandwidthUsed = 0;

// spinlock primitives
DEFINE_SPINLOCK( byte_send_lock);
DEFINE_SPINLOCK( byte_recv_lock);

// declaration of our function to be called by the kernel on a delay
static void timerBytesSent(struct work_struct* empty);
static void timerBytesRecvd(struct work_struct* empty);
static int findOutgoingGroup(struct sctp_chunk* chunkPtr);
static int findIncomingGroup(struct sctp_chunk* chunkPtr);
static void resetSendGroupLimits(void);
```

```
static void resetRecvGroupLimits(void);
void setMaxIncomingBandwidth(u64 bandwidthMax);
void purgeOutgoingStreamAssocGroups(void);
void purgeIncomingStreamAssocGroups(void);

// static macro which assigns our function to the time delay definition
DECLARE_DELAYED_WORK( SendWorkQ, timerBytesSent);
DECLARE_DELAYED_WORK( RecvWorkQ, timerBytesRecvd);

// initialize our bandwidth limiter
void initBandwidthLimiter(void)
{
        int i = 0;

        // this happens at startup so no lock required
        for (i = 0; i < MaxBandwidthGroups; i++)
        {
                // initialize the byte count and head pointer
                streamAssocSendGroups[i].byteCount = 0;
                streamAssocSendGroups[i].bandwidthDivisor = 0;
                streamAssocSendGroups[i].actualByteLimit = 0;
                streamAssocSendGroups[i].configuredByteLimit = 0;
                streamAssocSendGroups[i].rtxChunkCount = 0;
                streamAssocSendGroups[i].bytesDropped = 0;
                INIT_LIST_HEAD(&(streamAssocSendGroups[i].streamAssocList));

                streamAssocRecvGroups[i].byteCount = 0;
                streamAssocRecvGroups[i].bandwidthDivisor = 0;
                streamAssocRecvGroups[i].actualByteLimit = 0;
                streamAssocRecvGroups[i].configuredByteLimit = 0;
                streamAssocRecvGroups[i].rtxChunkCount = 0;
                streamAssocRecvGroups[i].bytesDropped = 0;
                INIT_LIST_HEAD(&(streamAssocRecvGroups[i].streamAssocList));
        }

        // reset limits
        resetSendGroupLimits();
        resetRecvGroupLimits();

        // log
        pr_info("%s:%d:\n", __func__, __LINE__);
}

void addOutgoingStreamAssocGroup(u16 groupId, u32 bwDivisor)
{
        // todo: some math to ensure the divisor is sane
        // spinlock primitives
        unsigned long flags;
        u32 rem = 0;

        // lock
        spin_lock_irqsave(&byte_send_lock, flags);

        // the bandwidth divisor marks if the grouping is active
        streamAssocSendGroups[groupId].bandwidthDivisor = div_u64_rem(100 * FP_MULT,
                        bwDivisor, &rem);

        // configure the byte limit
        streamAssocSendGroups[groupId].configuredByteLimit = div_u64_rem(
                        MaxBandwidthSend, streamAssocSendGroups[groupId].bandwidthDivisor,
                        &rem);

        // set the calculated byte limit to the configured limit
        streamAssocSendGroups[groupId].actualByteLimit =
                        streamAssocSendGroups[groupId].configuredByteLimit;

        // unlock
        spin_unlock_irqrestore(&byte_send_lock, flags);

        // log
        pr_info("%s:%d:groupId:%d:bwDivisor:%d:byteLimit:%llu\n", __func__,
                        __LINE__, groupId, streamAssocSendGroups[groupId].bandwidthDivisor,
                        streamAssocSendGroups[groupId].actualByteLimit);
}

void addIncomingStreamAssocGroup(u16 groupId, u32 bwDivisor)
{
        // todo: some math to ensure the divisor is sane
        // spinlock primitives
        unsigned long flags;
        u32 rem = 0;

        // lock
        spin_lock_irqsave(&byte_recv_lock, flags);

        // calculate the divisor for the percentage given by the user
        streamAssocRecvGroups[groupId].bandwidthDivisor = div_u64_rem(100 * FP_MULT,
                        bwDivisor, &rem);

        // now calculate the actual byte limit
        streamAssocRecvGroups[groupId].configuredByteLimit = div_u64_rem(
                        MaxBandwidthRecv, streamAssocRecvGroups[groupId].bandwidthDivisor,
                        &rem);

        // set the calculated byte limit to the configured limit
        streamAssocRecvGroups[groupId].actualByteLimit =
                        streamAssocRecvGroups[groupId].configuredByteLimit;

        // unlock
```

```
        spin_unlock_irqrestore(&byte_recv_lock, flags);

        // log
        pr_info("%s:%d:groupId:%d:bwDivisor:%d:byteLimit:%llu\n", __func__,
                        __LINE__, groupId, streamAssocRecvGroups[groupId].bandwidthDivisor,
                        streamAssocRecvGroups[groupId].actualByteLimit);
}

void purgeOutgoingStreamAssocGroups(void)
{
        // spinlock primitives
        unsigned long flags, i;

        // using the list
        struct list_head *ptr, *safePtr;

        // temporary pointer
        struct StreamAssoc* streamAssocPtr;

        // lock
        spin_lock_irqsave(&byte_send_lock, flags);

        // look for the definition of this association-stream
        for (i = 0; i < MaxBandwidthGroups; i++)
        {
                // in each group, iterate through the group list
                if (streamAssocSendGroups[i].bandwidthDivisor > 0)
                {
                        list_for_each_safe(ptr, safePtr,
                                        &(streamAssocSendGroups[i].streamAssocList))
                        {
                                streamAssocPtr = list_entry(ptr, struct StreamAssoc, list);
                                list_del(ptr);
                                kfree(streamAssocPtr);
                        }
                }
        }

        // unlock
        spin_unlock_irqrestore(&byte_send_lock, flags);

        // log
        pr_info("%s:%d", __func__, __LINE__);
}

void purgeIncomingStreamAssocGroups(void)
{
        // spinlock primitives
        unsigned long flags, i;

        // using the list
        struct list_head *ptr, *safePtr;

        // temporary pointer
        struct StreamAssoc* streamAssocPtr;

        // lock
        spin_lock_irqsave(&byte_recv_lock, flags);

        // look for the definition of this association-stream
        for (i = 0; i < MaxBandwidthGroups; i++)
        {
                // in each group, iterate through the group list
                if (streamAssocRecvGroups[i].bandwidthDivisor > 0)
                {
                        list_for_each_safe(ptr, safePtr,
                                        &(streamAssocRecvGroups[i].streamAssocList))
                        {
                                streamAssocPtr = list_entry(ptr, struct StreamAssoc, list);
                                list_del(ptr);
                                kfree(streamAssocPtr);
                        }
                }
        }

        // unlock
        spin_unlock_irqrestore(&byte_recv_lock, flags);

        // log
        pr_info("%s:%d", __func__, __LINE__);
}

// could this just be socket and SID? maybe
void addStreamAssocOutgoing(u16 groupId, struct sctp_association* assocPtr)
{
        // spinlock primitives
        unsigned long flags;

        // allocate the memory
        struct StreamAssoc* streamAssocPtr = (struct StreamAssoc*) kmalloc(
                        sizeof(struct StreamAssoc), GFP_KERNEL);

        // initialize the association
        streamAssocPtr->assocPtr = assocPtr;

        // lock
        spin_lock_irqsave(&byte_send_lock, flags);

        // now add it to the tail of our head pointer
```

```
        list_add_tail(&(streamAssocPtr->list),
                        &(streamAssocSendGroups[groupId].streamAssocList));

        // added new association, reset all groups
        resetSendGroupLimits();

        // unlock
        spin_unlock_irqrestore(&byte_send_lock, flags);

        // log
        pr_info("%s:%d:groupId:%d:assocPtr:0x%x\n", __func__, __LINE__, groupId,
                        (unsigned int) assocPtr);
}

void addStreamAssocIncoming(u16 groupId, struct sctp_association* assocPtr)
{
        // spinlock primitives
        unsigned long flags;

        // allocate the memory
        struct StreamAssoc* streamAssocPtr = (struct StreamAssoc*) kmalloc(
                        sizeof(struct StreamAssoc), GFP_KERNEL);

        // initialize the association
        streamAssocPtr->assocPtr = assocPtr;

        // lock
        spin_lock_irqsave(&byte_recv_lock, flags);

        // now add it to the tail of our head pointer
        list_add_tail(&(streamAssocPtr->list),
                        &(streamAssocRecvGroups[groupId].streamAssocList));

        // added new association, reset all groups
        resetRecvGroupLimits();

        // unlock
        spin_unlock_irqrestore(&byte_recv_lock, flags);

        // log
        pr_info("%s:%d:groupId:%d:assocPtr:0x%x\n", __func__, __LINE__, groupId,
                        (unsigned int) assocPtr);
}

// our timer hander
struct TimerReport
{
        u64 byteCount;
        u64 actualByteLimit;
        u64 configuredByteLimit;
        u64 rtxCount;
        u64 byteDropCount;
};

void timerBytesSent(struct work_struct* empty)
{
        // create array of byte counts
        struct TimerReport sendArrayTotals[MaxBandwidthGroups];

        // spinlock primitives
        unsigned long flags;
        int i;
        bool byteLimitChanged = false;

        // total for all groups
        u64 total_bytes_sent = 0;

        // lock
        spin_lock_irqsave(&byte_send_lock, flags);

        // for each group go through and perform group timer processing
        for (i = 0; i < MaxBandwidthGroups; i++)
        {
                // divisor greater than zero tells us this group is in use
                if (streamAssocSendGroups[i].bandwidthDivisor > 0)
                {
                        // is the last scan's byteCount + padding less than the
                        // actual byte limit?
                        if ((streamAssocSendGroups[i].byteCount + SCTP_DEFAULT_MINWINDOW)
                                        <= streamAssocSendGroups[i].actualByteLimit)
                        {
                                // mark for logging
                                byteLimitChanged = true;

                                // we didn't use all our group bandwidth in the last scan. give some up
                                // for other groups to use if they need it
                                streamAssocSendGroups[i].actualByteLimit -=
                                                SCTP_DEFAULT_MINWINDOW;

                                // give to the bandwidth free pool
                                SendBandwidthFreePool += SCTP_DEFAULT_MINWINDOW;
                        }

                        // save off the byte count for reporting
                        sendArrayTotals[i].actualByteLimit = streamAssocSendGroups[i].actualByteLimit;
                        sendArrayTotals[i].byteCount = streamAssocSendGroups[i].byteCount;
                        sendArrayTotals[i].rtxCount = streamAssocSendGroups[i].rtxChunkCount;
                        sendArrayTotals[i].byteDropCount = streamAssocSendGroups[i].bytesDropped;
                        total_bytes_sent += sendArrayTotals[i].byteCount;
```

```
                        // clear out periodic statistics
                        streamAssocSendGroups[i].byteCount = 0;
                        streamAssocSendGroups[i].rtxChunkCount = 0;
                        streamAssocSendGroups[i].bytesDropped = 0;
                }
        }

        // clear out the total bandwidth used counter
        SendBandwidthUsed = 0;

        // unlock
        spin_unlock_irqrestore(&byte_send_lock, flags);

        if ((total_bytes_sent > 0) || (true == byteLimitChanged))
        {
                pr_info(
                        "%s:%d:total_sent:%llu:send_free_pool:%llu:out0:%llu:out1:%llu:out2:%llu:out3
                             :%llu:bd0:%llu:bd1:%llu:bd2:%llu:bd3:%llu:rtx0:%llu:rtx1:%llu:rtx2:%llu
                             :rtx3:%llu",
                        __func__, __LINE__, total_bytes_sent, SendBandwidthFreePool,
                        sendArrayTotals[0].byteCount, sendArrayTotals[1].byteCount,
                        sendArrayTotals[2].byteCount, sendArrayTotals[3].byteCount,
                        sendArrayTotals[0].byteDropCount, sendArrayTotals[1].byteDropCount,
                        sendArrayTotals[2].byteDropCount, sendArrayTotals[3].byteDropCount,
                        sendArrayTotals[0].rtxCount, sendArrayTotals[1].rtxCount,
                        sendArrayTotals[2].rtxCount, sendArrayTotals[3].rtxCount);
        }

        // reschedule
        schedule_delayed_work(&SendWorkQ, msecs_to_jiffies(1000));
}

// report bytes received
void timerBytesRecvd(struct work_struct* empty)
{
        // create array of byte counts
        struct TimerReport recvArrayTotals[MaxBandwidthGroups];

        // spinlock primitives
        unsigned long flags;
        bool byteLimitChanged = false;
        int i;

        // total for all groups
        u64 total_bytes_recvd = 0;

        // lock
        spin_lock_irqsave(&byte_recv_lock, flags);

        // for each group go through and perform group timer processing
        for (i = 0; i < MaxBandwidthGroups; i++)
        {
                // divisor greater than zero tells us this group is in use
                if (streamAssocRecvGroups[i].bandwidthDivisor > 0)
                {
                        // is the last scan's byteCount + padding less than the
                        // actual byte limit?
                        if ((streamAssocRecvGroups[i].byteCount + SCTP_DEFAULT_MINWINDOW)
                                        <= streamAssocRecvGroups[i].actualByteLimit)
                        {
                                // mark for logging
                                byteLimitChanged = true;

                                // we didn't use all our group bandwidth in the last scan. give some up
                                // for other groups to use if they need it
                                streamAssocRecvGroups[i].actualByteLimit -=
                                                SCTP_DEFAULT_MINWINDOW;

                                // give to the bandwidth free pool
                                RecvBandwidthFreePool += SCTP_DEFAULT_MINWINDOW;
                        }

                        // save off the byte count for reporting
                        recvArrayTotals[i].actualByteLimit = streamAssocRecvGroups[i].actualByteLimit;
                        recvArrayTotals[i].byteCount = streamAssocRecvGroups[i].byteCount;
                        recvArrayTotals[i].rtxCount = streamAssocRecvGroups[i].rtxChunkCount;
                        recvArrayTotals[i].byteDropCount = streamAssocRecvGroups[i].bytesDropped;
                        total_bytes_recvd += recvArrayTotals[i].byteCount;

                        // clear out periodic statistics
                        streamAssocRecvGroups[i].byteCount = 0;
                        streamAssocRecvGroups[i].rtxChunkCount = 0;
                        streamAssocRecvGroups[i].bytesDropped = 0;
                }
        }

        // clear out the total bandwidth used counter
        RecvBandwidthUsed = 0;

        // unlock
        spin_unlock_irqrestore(&byte_recv_lock, flags);

        if ((total_bytes_recvd > 0) || (true == byteLimitChanged))
        {
                pr_info(
                        "%s:%d:total_recv:%llu:recv_free_pool:%llu:in0:%llu:in1:%llu:in2:%llu:in3:%
                             llu:bd0:%llu:bd1:%llu:bd2:%llu:bd3:%llu:rtx0:%llu:rtx1:%llu:rtx2:%llu:
                             rtx3:%llu",
```

```
                                __func__, __LINE__, total_bytes_recvd, RecvBandwidthFreePool,
                                recvArrayTotals[0].byteCount, recvArrayTotals[1].byteCount,
                                recvArrayTotals[2].byteCount, recvArrayTotals[3].byteCount,
                                recvArrayTotals[0].byteDropCount, recvArrayTotals[1].byteDropCount,
                                recvArrayTotals[2].byteDropCount, recvArrayTotals[3].byteDropCount,
                                recvArrayTotals[0].rtxCount, recvArrayTotals[1].rtxCount,
                                recvArrayTotals[2].rtxCount, recvArrayTotals[3].rtxCount );
        }

        // reschedule
        schedule_delayed_work(&RecvWorkQ, msecs_to_jiffies(1000));

}

// these functions decide if the caller shoud drop the chunk or let it pass through
// it also increments the received byte count up to the drop point
bool incomingChunk(struct sctp_chunk* chunkPtr)
{
        // spinlock primitives
        unsigned long flags = 0;
        __u16 chunkLen = 0;
        int groupId = -1;

        // return value
        bool dropChunk = false;

        // lock
        spin_lock_irqsave(&byte_recv_lock, flags);

        // we only care if we're dealing with a data chunk
        // we let all other chunk types through. this
        // allows for other chunks (like heartbeat) to
        // go through keeping the connection alive.
        if (chunkPtr->chunk_hdr->type == SCTP_CID_DATA)
        {
                // get chunk size
                chunkLen = ntohs(chunkPtr->chunk_hdr->length) - DATA_CHUNK_HDR_LEN;

                // find our group
                groupId = findIncomingGroup(chunkPtr);

                if (groupId >= 0)
                {
                        // first we check if this was a retransmitted chunk, and increment
                        // the retransmitted chunk counter for statistics logging
                        if ( chunkPtr->chunk_hdr->flags & SCTP_DATA_RTX )
                        {
                                // increment the received retransmitted chunk count
                                streamAssocRecvGroups[groupId].rtxChunkCount++;
                        }

                        // check if the used bandwidth has exceeded the total
                        // bandwidth count allocated
                        if ((RecvBandwidthUsed + chunkLen) < ReqMaxBandwidthRecv)
                        {
                                // we have not exceeded the total allowed bandwidth
                                // check if we have hit our actual limit
                                if ((streamAssocRecvGroups[groupId].byteCount + chunkLen)
                                                > streamAssocRecvGroups[groupId].actualByteLimit)
                                {
                                        // we're a group that tried to go past it's
                                        // actual limit. Are there any bytes left?
                                        if (chunkLen <= RecvBandwidthFreePool)
                                        {
                                                // yes, lets raise the limit borrowing from the free pool
                                                streamAssocRecvGroups[groupId].actualByteLimit +=
                                                                chunkLen;
                                                RecvBandwidthFreePool -= chunkLen;

                                                // account for this chunk in the total bandwidth used
                                                RecvBandwidthUsed += chunkLen;

                                                // We add this usage to the byte count and let pass
                                                streamAssocRecvGroups[groupId].byteCount += chunkLen;
                                        }
                                        else
                                        {
                                                // there is no more free bandwidth available. Are we a "
                                                     depressed" group
                                                if (streamAssocRecvGroups[groupId].actualByteLimit
                                                                < streamAssocRecvGroups[groupId].
                                                                     configuredByteLimit)
                                                {
                                                        // no more space left. We will now reclaim our limit
                                                             since
                                                        // the guarantee is against the configured limit.
                                                             this
                                                        // requires us to reset all the other groups as well.
                                                        resetRecvGroupLimits();

                                                        // even with reset limits we have to check if we have
                                                              enough
                                                        // bandwidth to send the chunk up the stack, so we
                                                             check again
                                                        if ((streamAssocRecvGroups[groupId].byteCount
                                                                        + chunkLen)
                                                                        > streamAssocRecvGroups[groupId].
                                                                             actualByteLimit)
                                                        {
```

```
                                                // we just don't have the bandwidth. drop.
                                                dropChunk = true;
                                                pr_debug(
                                                        "%s:%d:Drop, Depressed:
                                                             groupId:%d:byteCount:%
                                                             llu:chunkLen:%d:aBL:%
                                                             llu:RFP:%llu",
                                                        __func__, __LINE__, groupId,
                                                        streamAssocRecvGroups[groupId
                                                             ].byteCount,
                                                        chunkLen,
                                                        streamAssocRecvGroups[groupId
                                                             ].actualByteLimit,
                                                        RecvBandwidthFreePool);
                                        }
                                        else
                                        {
                                                // we have available bandwidth now!
                                                // We add this usage to the byte count and
                                                     let pass
                                                streamAssocRecvGroups[groupId].byteCount +=
                                                        chunkLen;

                                                // add to the total received bandwidth used
                                                RecvBandwidthUsed += chunkLen;
                                        }
                                }
                                else
                                {
                                        // we're not a depressed group,
                                        // and no excess bandwidth available,
                                        // so drop
                                        dropChunk = true;
                                        pr_debug(
                                                "%s:%d:Drop, Not Depressed, No Excess
                                                     ::groupId:%d:byteCount:%llu:
                                                     chunkLen:%d:aBL:%llu:RFP:%llu",
                                                __func__, __LINE__, groupId,
                                                streamAssocRecvGroups[groupId].
                                                     byteCount,
                                                chunkLen,
                                                streamAssocRecvGroups[groupId].
                                                     actualByteLimit,
                                                RecvBandwidthFreePool);
                                }
                        }
                }
                else
                {
                        // we still have room, allow to pass
                        streamAssocRecvGroups[groupId].byteCount += chunkLen;

                        // add to the total received bandwidth used
                        RecvBandwidthUsed += chunkLen;
                }
        }
        else
        {
                // oops we hit the ceiling. we need to reset all our groups
                // and drop this chunk
                resetRecvGroupLimits();
                dropChunk = true;
                pr_debug(
                        "%s:%d:Drop, Max Hit:RecvBandwidthUsed:%llu:chunkLen:%d:
                             MaxBandwidthRecv:%llu:RecvFreePool:%llu",
                        __func__, __LINE__, RecvBandwidthUsed, chunkLen,
                        ReqMaxBandwidthRecv, RecvBandwidthFreePool);
        }

        if ( true == dropChunk )
        {
                streamAssocRecvGroups[groupId].bytesDropped += chunkLen;
        }

        // if we're about to run out of bandwidth (say within 2*PMTU)
        // or we actually ran out then we need to raise the congestion
        // notification flag
        if ((RecvBandwidthUsed + (chunkPtr->asoc->pathmtu * 2)) >= ReqMaxBandwidthRecv)
        {
                // we're about to hit the limit. raise the flag!
                chunkPtr->asoc->need_ecne = 1;
        }
        else
        {
                // clear the flag
                chunkPtr->asoc->need_ecne = 0;
        }
    }
}

// unlock
spin_unlock_irqrestore(&byte_recv_lock, flags);

// return back drop value
return dropChunk;
}

// these functions decide if the caller shoud drop the chunk or let it pass through
// it also incrememnts the received byte count up to the drop point
```

```
bool outgoingChunk(struct sctp_chunk* chunkPtr)
{
        // spinlock primitives
        unsigned long flags = 0;
        __u16 chunkLen = 0;
        int groupId = -1;

        // return value
        bool dropChunk = false;

        // lock
        spin_lock_irqsave(&byte_send_lock, flags);

        // we only care if we're dealing with a data chunk
        // we let all other chunk types through. this
        // allows for other chunks (like heartbeat) to
        // go through keeping the connection alive.
        if (chunkPtr->chunk_hdr->type == SCTP_CID_DATA)
        {
                // get chunk size
                chunkLen = ntohs(chunkPtr->chunk_hdr->length) - DATA_CHUNK_HDR_LEN;

                // find our group
                groupId = findOutgoingGroup(chunkPtr);

                if (groupId >= 0)
                {
                        // first check to see if this chunk is a retransmission
                        // if so, mark our NEW FLAG to tell the receiver
                        if ( chunkPtr->resent > 0 )
                        {
                                // mark this chunk as a rtx chunk
                                chunkPtr->chunk_hdr->flags |= SCTP_DATA_RTX;

                                // increment the received retransmitted chunk count
                streamAssocSendGroups[groupId].rtxChunkCount++;
                        }

                        // check if the used bandwidth has exceeded the total
                        // bandwidth count allocated
                        if ((SendBandwidthUsed + chunkLen) < ReqMaxBandwidthSend)
                        {
                                // we have not exceeded the total allowed bandwidth
                                // check if we have hit our actual limit
                                if ((streamAssocSendGroups[groupId].byteCount + chunkLen)
                                                > streamAssocSendGroups[groupId].actualByteLimit)
                                {
                                        // we're a group that tried to go past it's
                                        // actual limit. Are there any bytes left?
                                        if (chunkLen <= SendBandwidthFreePool)
                                        {
                                                // yes, lets raise the limit borrowing from the free pool
                                                streamAssocSendGroups[groupId].actualByteLimit +=
                                                                chunkLen;
                                                SendBandwidthFreePool -= chunkLen;

                                                // account for this chunk in the total bandwidth used
                                                SendBandwidthUsed += chunkLen;

                                                // We add this usage to the byte count and let pass
                                                streamAssocSendGroups[groupId].byteCount += chunkLen;
                                        }
                                        else
                                        {
                                                // there is no more free bandwidth available. Are we a "
                                                   depressed" group
                                                if (streamAssocSendGroups[groupId].actualByteLimit
                                                                < streamAssocSendGroups[groupId].
                                                                     configuredByteLimit)
                                                {
                                                        // no more space left. We will now reclaim our limit
                                                           since
                                                        // the guarantee is against the configured limit.
                                                             this
                                                        // requires us to reset all the other groups as well.
                                                        resetSendGroupLimits();

                                                        // even with reset limits we have to check if we have
                                                             enough
                                                        // bandwidth to send the chunk up the stack, so we
                                                           check again
                                                        if ((streamAssocSendGroups[groupId].byteCount
                                                                        + chunkLen)
                                                                        > streamAssocSendGroups[groupId].
                                                                             actualByteLimit)
                                                        {
                                                                // we just don't have the bandwidth. drop.
                                                                dropChunk = true;

                                                                pr_debug(
                                                                                "%s:%d:Drop, Depressed:
                                                                                     groupId:%d:byteCount:%
                                                                                     llu:chunkLen:%d:aBL:%
                                                                                     llu:SFP:%llu",
                                                                __func__, __LINE__, groupId,
                                                                streamAssocSendGroups[groupId
                                                                     ].byteCount,
                                                                chunkLen,
                                                                streamAssocSendGroups[groupId
```

```
                                                                        ].actualByteLimit,
                                                                    SendBandwidthFreePool);
                                                }
                                                else
                                                {
                                                        // we have available bandwidth now!
                                                        // We add this usage to the byte count and
                                                            let pass
                                                        streamAssocSendGroups[groupId].byteCount +=
                                                                chunkLen;

                                                        // add to the total received bandwidth used
                                                        SendBandwidthUsed += chunkLen;
                                                }
                                        }
                                        else
                                        {
                                                // we're not a depressed group,
                                                // and no excess bandwidth available,
                                                // so drop
                                                dropChunk = true;

                                                pr_debug(
                                                        "%s:%d:Drop, Not Depressed, No Excess
                                                            ::groupId:%d:byteCount:%llu:
                                                            chunkLen:%d:aBL:%llu:SFP:%llu",
                                                        __func__, __LINE__, groupId,
                                                        streamAssocSendGroups[groupId].
                                                            byteCount,
                                                        chunkLen,
                                                        streamAssocSendGroups[groupId].
                                                            actualByteLimit,
                                                        SendBandwidthFreePool);
                                        }
                                }
                        }
                        else
                        {
                                // we still have room, allow to pass
                                streamAssocSendGroups[groupId].byteCount += chunkLen;

                                // add to the total received bandwidth used
                                SendBandwidthUsed += chunkLen;
                        }
                }
                else
                {
                        // oops we hit the ceiling. we need to reset all our groups
                        // and drop this chunk
                        resetSendGroupLimits();
                        dropChunk = true;

                        pr_debug(
                                "%s:%d:Drop, Max Hit:SendBandwidthUsed:%llu:chunkLen:%d:
                                    MaxBandwidthSend:%llu:SendFreePool:%llu",
                                __func__, __LINE__, SendBandwidthUsed, chunkLen,
                                ReqMaxBandwidthSend, SendBandwidthFreePool);
                }

                if ( true == dropChunk )
                {
                        streamAssocSendGroups[groupId].bytesDropped += chunkLen;
                }

                // if we're about to run out of bandwidth (say within 2*PMTU)
                // or we actually ran out then we need to raise the congestion
                // notification flag
                if ((SendBandwidthUsed + (chunkPtr->asoc->pathmtu * 2)) >= ReqMaxBandwidthSend)
                {
                        // we're about to hit the limit. raise the flag!
                        chunkPtr->need_ce = 1;
                }
                else
                {
                        // clear the flag
                        chunkPtr->need_ce = 0;
                }
        }
    }

    // unlock
    spin_unlock_irqrestore(&byte_send_lock, flags);

    // return back drop value
    return dropChunk;
}

void setMaxOutgoingBandwidth(u64 bandwidthMax)
{
    // todo: some math to ensure the divisor is sane
    // spinlock primitives
    unsigned long flags;

    // lock
    spin_lock_irqsave(&byte_send_lock, flags);

    // save off the original request
    ReqMaxBandwidthSend = bandwidthMax;
```

```
        // set the maximum bandwidth allowed for outgoing chunks
        MaxBandwidthSend = bandwidthMax * FP_MULT;

        // we just changed the ceiling height. reset all groups
        resetSendGroupLimits();

        // unlock
        spin_unlock_irqrestore(&byte_send_lock, flags);

        // log
        pr_info("%s:%d:%llu\n", __func__, __LINE__, MaxBandwidthSend);

}

void setMaxIncomingBandwidth(u64 bandwidthMax)
{
        // todo: some math to ensure the divisor is sane
        // spinlock primitives
        unsigned long flags;

        // lock
        spin_lock_irqsave(&byte_recv_lock, flags);

        // save off original request
        ReqMaxBandwidthRecv = bandwidthMax;

        // set the maximum bandwidth allowed for outgoing chunk
        MaxBandwidthRecv = bandwidthMax * FP_MULT;

        // we just changed the ceiling height. reset all groups
        resetRecvGroupLimits();

        // unlock
        spin_unlock_irqrestore(&byte_recv_lock, flags);

        // log
        pr_info("%s:%d:%llu\n", __func__, __LINE__, MaxBandwidthRecv);

}

// helper function to find the partition that corresponds
// to the passed in chunk. WE ASSUME LOCKING HAS BEEN
// TAKEN CARE OF!
static int findOutgoingGroup(struct sctp_chunk* chunkPtr)
{
        unsigned int i = 0;
        struct list_head* ptr;
        struct StreamAssoc* entry;

        // return value
        int groupId = -1;

        // look for the definition of this association-stream
        for (i = 0; i < MaxBandwidthGroups; i++)
        {
                // in each group, iterate through the group list
                // to find this chunk's stream-assoc. I'm sure
                // there's a more efficient way to do this...
                if (streamAssocSendGroups[i].bandwidthDivisor > 0)
                {
                        list_for_each(ptr, &(streamAssocSendGroups[i].streamAssocList))
                        {
                                entry = list_entry(ptr, struct StreamAssoc, list);
                                if ((0 != entry) && (entry->assocPtr == chunkPtr->asoc))
                                {
                                        // we found our stream-association in this group.
                                        groupId = i;

                                        // end the loop
                                        goto finish;
                                }
                        }
                }
        }

        finish:

        return groupId;
}

static int findIncomingGroup(struct sctp_chunk* chunkPtr)
{
        unsigned int i = 0;
        struct list_head* ptr;
        struct StreamAssoc* entry;

        // return value
        int groupId = -1;

        // look for the definition of this association-stream
        for (i = 0; i < MaxBandwidthGroups; i++)
        {
                // in each group, iterate through the group list
                // to find this chunk's stream-assoc. I'm sure
                // there's a more efficient way to do this...
                if (streamAssocRecvGroups[i].bandwidthDivisor > 0)
                {
                        list_for_each(ptr, &(streamAssocRecvGroups[i].streamAssocList))
                        {
```

```
                                entry = list_entry(ptr, struct StreamAssoc, list);
                                if ((0 != entry) && (entry->assocPtr == chunkPtr->asoc))
                                {
                                        // we found our stream-association in this group.
                                        groupId = i;

                                        // end the loop
                                        goto finish;
                                }
                        }
                }
        }

        finish:

        return groupId;

}

// it is expected that the lock is already taken out
// before calling this function.
void resetSendGroupLimits(void)
{
        unsigned int i = 0;
        u64 totalConfigured = 0;

        // create array of byte counts
        struct TimerReport sendArrayTotals[MaxBandwidthGroups];

        // look for the definition of this association-stream
        for (i = 0; i < MaxBandwidthGroups; i++)
        {
                // check if this group is active
                if (streamAssocSendGroups[i].bandwidthDivisor > 0)
                {
                        // save off for logging
                        sendArrayTotals[i].actualByteLimit =
                                        streamAssocSendGroups[i].actualByteLimit;
                        sendArrayTotals[i].configuredByteLimit =
                                        streamAssocSendGroups[i].configuredByteLimit;
                        sendArrayTotals[i].byteCount = streamAssocSendGroups[i].byteCount;

                        // reset this groups actual limit to the configured limit
                        streamAssocSendGroups[i].actualByteLimit =
                                        streamAssocSendGroups[i].configuredByteLimit;

                        // tally up the total configured bytes
                        totalConfigured += streamAssocRecvGroups[i].configuredByteLimit;
                }
        }

        // log the data before reset
        pr_info(
                        "%s:%d:actl0:%llu:confl0:%llu:bc0:%llu:actl1:%llu:confl1:%llu:bc1:%llu:actl2:%llu:
                             confl2:%llu:bc2:%llu:actl3:%llu:confl3:%llu:bc3:%llu:RBU:%llu:RBF:%llu\n",
                        __func__, __LINE__, sendArrayTotals[0].actualByteLimit,
                        sendArrayTotals[0].configuredByteLimit,
                        sendArrayTotals[0].byteCount, sendArrayTotals[1].actualByteLimit,
                        sendArrayTotals[1].configuredByteLimit,
                        sendArrayTotals[1].byteCount, sendArrayTotals[2].actualByteLimit,
                        sendArrayTotals[2].configuredByteLimit,
                        sendArrayTotals[2].byteCount, sendArrayTotals[3].actualByteLimit,
                        sendArrayTotals[3].configuredByteLimit,
                        sendArrayTotals[3].byteCount, SendBandwidthUsed,
                        SendBandwidthFreePool);

        // set the receive bandwidth free pool to zero
        SendBandwidthFreePool = 0;
}

// it is expected that the lock is already taken out
// before calling this function.
void resetRecvGroupLimits(void)
{
        unsigned int i = 0;
        u64 totalConfigured = 0;

        // create array of byte counts
        struct TimerReport recvArrayTotals[MaxBandwidthGroups];

        // look for the definition of this association-stream
        for (i = 0; i < MaxBandwidthGroups; i++)
        {
                // check if this group is active
                if (streamAssocRecvGroups[i].bandwidthDivisor > 0)
                {
                        // save off for logging
                        recvArrayTotals[i].actualByteLimit =
                                        streamAssocRecvGroups[i].actualByteLimit;
                        recvArrayTotals[i].configuredByteLimit =
                                        streamAssocRecvGroups[i].configuredByteLimit;
                        recvArrayTotals[i].byteCount = streamAssocRecvGroups[i].byteCount;

                        // reset this groups actual limit to the configured limit
                        streamAssocRecvGroups[i].actualByteLimit =
                                        streamAssocRecvGroups[i].configuredByteLimit;

                        // tally up the total configured bytes
                        totalConfigured += streamAssocRecvGroups[i].configuredByteLimit;
```

```
                }
        }

        // log the data before reset
        pr_info(
                        "%s:%d:actl0:%llu:confl0:%llu:bc0:%llu:actl1:%llu:confl1:%llu:bc1:%llu:actl2:%llu:
                             confl2:%llu:bc2:%llu:actl3:%llu:confl3:%llu:bc3:%llu:RBU:%llu:RBF:%llu\n",
                        __func__, __LINE__, recvArrayTotals[0].actualByteLimit,
                        recvArrayTotals[0].configuredByteLimit,
                        recvArrayTotals[0].byteCount, recvArrayTotals[1].actualByteLimit,
                        recvArrayTotals[1].configuredByteLimit,
                        recvArrayTotals[1].byteCount, recvArrayTotals[2].actualByteLimit,
                        recvArrayTotals[2].configuredByteLimit,
                        recvArrayTotals[2].byteCount, recvArrayTotals[3].actualByteLimit,
                        recvArrayTotals[3].configuredByteLimit,
                        recvArrayTotals[3].byteCount, RecvBandwidthUsed,
                        RecvBandwidthFreePool);

        // set the receive bandwidth free pool to zero
        RecvBandwidthFreePool = 0;
}
```

REFERENCES

Arch linux. http://www.archlinux.org. Accessed: 2014.

Debian – the universal operating system. http://www.debian.org. Accessed: 2014.

The linux kernel archive. http://www.kernel.org. Accessed: 2014.

The xen project, the powerful open source industry standard for virtualization. http://www.xen.org. Accessed: 2014.

Akella, A. V. and Xiong, K. (2014). Quality of service (qos)-guaranteed network resource allocation via software defined networking (sdn). In *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on*, pages 7–13. IEEE.

Amamou, A., Bourguiba, M., Haddadou, K., and Pujolle, G. (2012). A dynamic bandwidth allocator for virtual machines in a cloud environment. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pages 99–104. IEEE.

Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. (2011). Towards predictable datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 242–253. ACM.

Ballani, H., Jang, K., Karagiannis, T., Kim, C., Gunawardena, D., and O'Shea, G. (2013). Chatty tenants and the cloud network sharing problem. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 171–184.

Bardgett, J. and Zou, C. (2012). nswitching: Virtual machine aware relay hardware switching to improve intra-nic virtual machine traffic. In *Communications (ICC), 2012 IEEE International Conference on*, pages 2700–2705. IEEE.

Cherkasova, L. and Gardner, R. (2005). Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In *USENIX Annual Technical Conference, General Track*, volume 50.

Chinni, S. and Hiremane, R. (2007). Virtual machine device queues. *White paper, Intel Corporation.*

Divakaran, D. M. and Gurusamy, M. (2015). Towards flexible guarantees in clouds: Adaptive bandwidth allocation and pricing. *Parallel and Distributed Systems, IEEE Transactions on*, 26(6):1754–1764.

Dong, Y., Dai, J., Huang, Z., Guan, H., Tian, K., and Jiang, Y. (2009). Towards high-quality i/o virtualization. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, page 12. ACM.

Dong, Y., Yang, X., Li, J., Liao, G., Tian, K., and Guan, H. (2012). High performance network virtualization with sr-iov. *Journal of Parallel and Distributed Computing*, 72(11):1471–1480.

Emmerich, P., Raumer, D., Wohlfart, F., and Carle, G. (2014). Performance characteristics of virtual switching. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 120–125. IEEE.

Gamage, S., Kangarlou, A., Kompella, R. R., and Xu, D. (2011). Opportunistic flooding to improve tcp transmit performance in virtualized clouds. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 24. ACM.

Guo, C., Lu, G., Wang, H. J., Yang, S., Kong, C., Sun, P., Wu, W., and Zhang, Y. (2010). Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International COnference*, page 15. ACM.

Kartik, N. (2003). Tcp optimized for short flows.

Lee, J., Lee, M., Popa, L., Turner, Y., Banerjee, S., Sharma, P., and Stephenson, B. (2013). Cloudmirror: Application-aware bandwidth reservations in the cloud. In *HotCloud*. Citeseer.

Lee, J., Turner, Y., Lee, M., Popa, L., Banerjee, S., Kang, J.-M., and Sharma, P. (2014). Application-driven bandwidth guarantees in datacenters. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 467–478. ACM.

Li, A., Yang, X., Kandula, S., and Zhang, M. (2010). Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM.

Liu, Y., Xhagjika, V., Vlassov, V., and Al Shishtawy, A. (2014). Bwman: Bandwidth manager for elastic services in the cloud. In *Parallel and Distributed Processing with Applications (ISPA), 2014 IEEE International Symposium on*, pages 217–224. IEEE.

Luglio, M., Roseti, C., and Zampognaro, F. (2014). Transport layer optimization for cloud computing applications via satellite: Tcp noordwijk+. *Communications, China*, 11(12):105–119.

Luo, Y. (2010). Network i/o virtualization for cloud computing. *IT Professional Magazine*, 12(5):36.

Luo, Y., Cascon, P., Murray, E., and Ortega, J. (2009). Accelerating openflow switching with network processors. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 70–71. ACM.

Menon, A., Santos, J. R., Turner, Y., Janakiraman, G. J., and Zwaenepoel, W. (2005). Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 13–23. ACM.

Menon, A. and Zwaenepoel, W. (2008). Optimizing tcp receive performance. In *USENIX Annual Technical Conference*, pages 85–98. Boston, MA.

Misra, S., Das, S., Khatua, M., and Obaidat, M. S. (2014). Qos-guaranteed bandwidth shifting and redistribution in mobile cloud environment. *Cloud Computing, IEEE Transactions on*, 2(2):181–193.

Mogul, J. C. and Popa, L. (2012). What we talk about when we talk about cloud network performance. *ACM SIGCOMM Computer Communication Review*, 42(5):44–48.

Pawar, C. S. and Wagh, R. B. (2012). Priority based dynamic resource allocation in cloud computing. In *Cloud and Services Computing (ISCOS), 2012 International Symposium on*, pages 1–6. IEEE.

Popa, L., Kumar, G., Chowdhury, M., Krishnamurthy, A., Ratnasamy, S., and Stoica, I. (2012). Faircloud: sharing the network in cloud computing. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 187–198. ACM.

Popa, L., Yalagandula, P., Banerjee, S., Mogul, J. C., Turner, Y., and Santos, J. R. (2013). Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 351–362. ACM.

Radhakrishnan, S., Pan, R., Vahdat, A., Varghese, G., et al. (2012). Netshare and stochastic netshare: predictable bandwidth allocation for data centers. *ACM SIGCOMM Computer Communication Review*, 42(3):5–11.

Ramakrishnan, K., Floyd, S., and Black, D. (2001). Rfc 3168. *The addition of Explicit Congestion Notification (ECN) to IP. The Internet Society*.

Rodrigues, H., Santos, J. R., Turner, Y., Soares, P., and Guedes, D. (2011). Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *WIOV*.

Scharf, M. and Kiesel, S. (2006). Nxg03-5: Head-of-line blocking in tcp and sctp: Analysis and measurements. In *Global Telecommunications Conference, 2006. GLOBECOM'06. IEEE*, pages 1–5. IEEE.

Shieh, A., Kandula, S., Greenberg, A. G., Kim, C., and Saha, B. (2011). Sharing the data center network. In *NSDI*.

Stewart, R. R. and Xie, Q. (2002). *Stream Control Transmission Protocol (SCTP): A Reference Guide*. Addison-Wesley.

Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2008). A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55.

Wang, J., Wright, K.-L., and Gopalan, K. (2008). Xenloop: a transparent high performance inter-vm network loopback. In *Proceedings of the 17th international symposium on High performance distributed computing*, pages 109–118. ACM.

Xiaojing, W., Wei, Y., Haowei, W., Linjie, D., and Chi, Z. (2012). Evaluation of traffic control in virtual environment. In *Distributed Computing and Applications to Business, Engineering & Science (DCABES), 2012 11th International Symposium on*, pages 332–335. IEEE.

Ye, G., Saadawi, T. N., and Lee, M. (2005). On explicit congestion notification for stream control transmission protocol in lossy networks. *Cluster Computing*, 8(2-3):147–156.

Zhang, J., Ren, F., Yue, X., Shu, R., and Lin, C. (2014). Sharing bandwidth by allocating switch buffer in data center networks. *Selected Areas in Communications, IEEE Journal on*, 32(1):39–51.

Zhang, X., McIntosh, S., Rohatgi, P., and Griffin, J. L. (2007). Xensocket: A high-throughput interdomain transport for virtual machines. In *Middleware 2007*, pages 184–203. Springer.

Zhu, J., Li, D., Wu, J., Liu, H., Zhang, Y., and Zhang, J. (2012). Towards bandwidth guarantee in multi-tenancy cloud computing networks. In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–10. IEEE.