Automatic Dynamic Run-time Optical Network Reservations

John R. Lange Ananth I. Sundararaj Peter A. Dinda
{jarusl,ais,pdinda}@cs.northwestern.edu
Department of Computer Science, Northwestern University

Abstract

Optical networking may dramatically change high performance distributed computing. One reason is that optical networks can support provisioning dynamically configurable lightpaths, a form of circuit switching, through reservations. However, to use it (and all other network reservation mechanisms), the user or developer must modify the application. We present a system, VRESERVE, that automatically and dynamically creates network reservation requests based on the inferred network demands of running distributed and/or parallel applications with no modification to the application or operating system, and no input from the user or developer. Our execution model is a collection of virtual machines interconnected by an overlay network. The overlay network infers application demands, providing a dynamic run-time assessment of the application's topology and traffic load matrix. We then reserve lightpaths corresponding to the topology and use the overlay to forward virtual network traffic over them. We evaluate our system on the OMNInet network.

1 Introduction

Many people believe that high speed optical networks will dramatically extend the capabilities of high performance distributed computing. This potential has prompted the creation of private national and international optical networks [19, 1, 20], and the development of new models for using them. For example, the OptIPuter project [24] uses a dedicated optical network to interconnect large compute centers, data storage farms, and visualization centers.

When used with the traditional packet switching paradigm, optical networks operate with extremely high bandwidth but also very high latency [3]. This observation has motivated the optical networking community to investigate supplementing packet switching with capabilities for optical circuit switching. Circuit switching is strongly tied to resource reservations. While considerable work has gone into reservation mechanisms for packet switched networks, there has been little deployment of these mechanisms because of concerns about state size on routers. Circuit switched networks provide an environment that is potentially more suitable for reservation mechanisms. Since the state required in a circuit switch is already per-connection, adding per-connection reservations incurs only a constant factor increase in state size. Network and CPU reservations are powerful mechanisms for *guaranteeing* stable application performance.

Circuit switching requires establishing specific paths and the attributes of those paths, e.g., such as throughput and latency. Currently, this places a new requirement on either the user, who must *manually* reserve the network on behalf of the application, or the developer, who must specifically call a reservation system API. In either case, manual intervention is required to determine what circuits are needed and how to provision them. To date very little work has been done on automatic network reservations based entirely on the application's needs at run time.

In this paper, we show that it is both feasible and relatively straightforward to *automatically* determine the necessary circuits and reserve them appropriately. Further, we can do so dynamically, changing circuits and reservations at run-time as the communication needs of the application change. Finally, our method, which is based on our own and others' virtualization technologies and our own inference mechanisms, works with *existing, unmodified applications and operating systems* with *no user or developer intervention*. There is good reason to believe that our work will readily extend to other network reservation schemes as well.

Our work takes place in the context of Virtuoso [22, 8], a system for grid computing using virtual machines. We leverage the following components of Virtuoso:

- Virtual Machine Monitors (VMM). We use VMware GSX Server [31] to create virtual machines that can

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, EIA-0224449, and gifts from VMware and Dell. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF), VMware, or Dell.

support any operating system. Our techniques are not tied to the use of VMware. We simply need access to the virtual network interface created by a VMM. Other options for creating virtual execution environments such as User-Mode Linux [4] and VServer [17] also provide this abstraction.

- VNET. VNET [27, 28] is an overlay networking tool we have developed that creates the illusion for the user's VMs that they are connected to the user's LAN, while supporting arbitrary overlay topologies, routing, and even overlay link types that provide the mechanisms needed to maximize overlay network performance for a given traffic pattern, network performance characteristics, and diverse security policies. [1]

- VTTIF (Virtual Traffic and Topology Inference Framework). VTTIF [12] observes every packet sent by a VM and infers from this traffic a global communication topology and traffic load matrix among a collection of VMs.

In earlier work [28, 29], we showed how we can use the traffic matrices inferred by VTTIF to drive the mechanisms of VNET (and VM migration) to dynamically adapt the locations and connectivity of VMs to improve application performance. In that work, we automatically adapted the unmodified application to the network. In this work, we automatically adapt the network to the unmodified application.

The idea of dynamically creating overlay networks has an analogue in the paradigm of creating optical channels between nodes. Instead of creating an overlay network on top of the existing Internet infrastructure, we request a dedicated light path from the optical network reservation system. For our system we experimented with ODIN [18], a set of optical network services, including provisioning capabilities, integrated into OMNInet [15], an experimental circuit switched optical network. We used VTTIF to monitor the application, and generated ODIN requests based on the inferred topology and traffic load matrix.

VRESERVE alleviates the reservation responsibility for both the user and the developer. In fact the environment experienced by both is exactly the same as when a network without reservations is used. By automatically requesting network reservations at run-time we have enabled any application to transparently and painlessly use dedicated high speed reservable networks to increase communication performance.

[1]VNET is available from virtuoso.cs.northwestern.edu.

2 Optical networks

Although optical networking has existed since the 1980s, there has been a recent resurgence of interest for the following reasons:

- Practical optical domain switching and amplification mechanisms have been developed, allowing the majority of a network path to be purely optical [3, 32].

- The throughput possible in existing optical fiber has been growing dramatically [21]. As there is much existing "dark fiber", this means that dedicated or shared wide area optical networks for high performance computing are becoming feasible [9, 23].

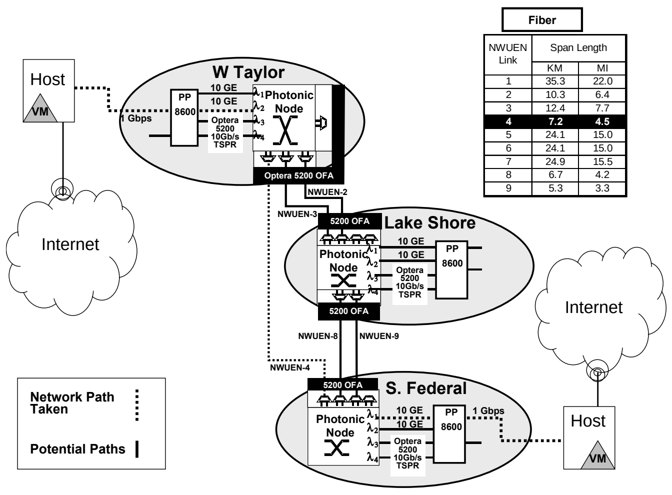
- Deployment of network reservation systems that can be used by an end user has stalled. The result is that commodity Internet performance has become increasingly unpredictable, even on dedicated IP networks.

The core of an optical network is built using optical amplifiers, optical switches, and interconnected using fiber-optic cables. Bits are injected into the network by modulating a laser beam feeding a cable. The center frequency of the beam is typically referred to as the "lambda". An optical amplifier increases the intensity of a range of frequencies. An optical switch directs light of one frequency (lambda) that arrives on an input port to some output port, possibly changing its frequency in the process. The mapping from input ports and frequencies to output ports and frequencies defines the configuration of the switch. By configuring interconnected switches appropriately, a light path can be established from a source host to a destination host. Such configuration is analogous to call setup in a circuit switched network.

Networks such as OMNInet [15], Canarie [1], and NetherLight [20] allow authorized entities to reserve and provision optical lightpaths. Beyond basic connectivity, this primitive can be used to create logically separate networks (perhaps for different groups) that share the same underlying physical resources. It could also be used directly by applications.

2.1 OMNInet and ODIN

In our evaluation, we use the OMNInet network and the ODIN light path reservation system. Figure 1 shows the physical topology of a section of OMNInet. OMNInet is an experimental fully connected network that spans several sites in Chicago. We use machines directly connected to the optical switches at two of the sites. OMNInet is run by the International Center for Advanced Internet Research (iCAIR).



NWUEN Link	Span Length	
	KM	MI
1	35.3	22.0
2	10.3	6.4
3	12.4	7.7
4	7.2	4.5
5	24.1	15.0
6	24.1	15.0
7	24.9	15.5
8	6.7	4.2
9	5.3	3.3

Figure 1. Physical topology of the OMNInet testbed network.

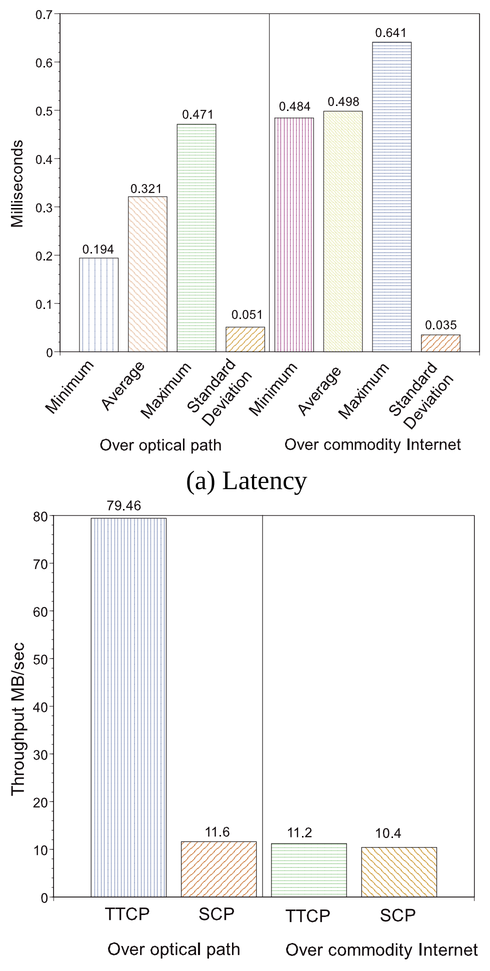
```
Reservation API:
CreatePath(<srcIP>, <dstIP>, <bw>, <lat>);
TeardownPath(<srcIP>, <dstIP>);

ODIN Interface:
oclient -c <srcIP> <dstIP> <lambda#> <flags>
oclient -t <pathID>
```

Figure 2. Reservation API.

ODIN is a lightpath provisioning system that iCAIR developed for use on OMNInet. Figure 2 shows the ODIN provisioning interface. ODIN's interface for establishing a lightpath is very similar to VNET's interface for creating an overlay link. ODIN uses IP addresses to identify network nodes. A path reservation request consists of the source and destination IP addresses, and the required long term average bandwidth and latency of the path. Networks are constructed by making a reservation request for each link. ODIN then uses Ethernet VLANs to ensure that the given network is fully restricted to the hosts in the network graph.

The time to configure the OMNInet network (create a collection of lightpaths) is rather large as the expectation is that this will be done infrequently. The average setup time we observed is ∼15 seconds, which includes updating all the switches. Figure 3 shows the ideal and measured (using ping and ttcp) latency and throughput of the path denoted on Figure 1 as compared to a path over the commodity Internet.



(a) Latency

(b) Throughput

Figure 3. Latency and throughput of the optical path as compared to the commodity Internet.

2.2 Reservations in other networks

It is important to note that our work is intended to generalize to other network reservation systems, whether they are optical or not. A unifying feature of network reservation systems is that they require the reserver to provide a model of the traffic that will be sent along a path and a specification of its latency and throughput requirements [6]. Our system provides this information, which could be used with, for example, GARA [14].

3 Automatic dynamic reservations

A high-level view of the system is shown in Figure 4. Each Ethernet packet sent by the application is diverted by

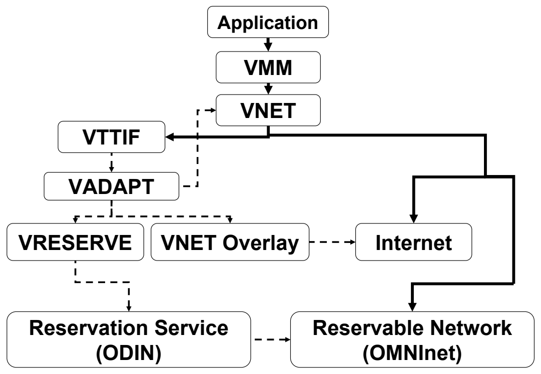


Figure 4. System overview. Dashed lines indicate control signals, solid lines denote actual network traffic.

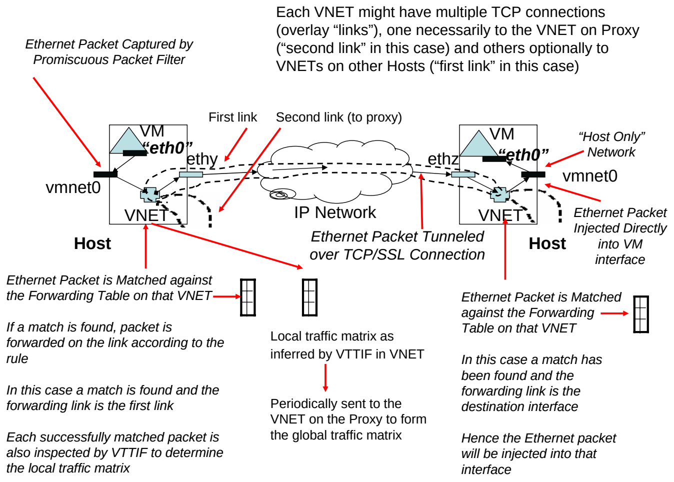

Each VNET might have multiple TCP connections (overlay "links"), one necessarily to the VNET on Proxy ("second link" in this case) and others optionally to VNETs on other Hosts ("first link" in this case)
Ethernet Packet Captured by Promiscuous Packet Filter
First link
Second link (to proxy)
VM
"eth0"
ethy
vmnet0
VNET
Host
IP Network
ethz
VM
"eth0"
"Host Only" Network
vmnet0
VNET
Host
Ethernet Packet Injected Directly into VM interface
Ethernet Packet Tunneled over TCP/SSL Connection
Ethernet Packet is Matched against the Forwarding Table on that VNET
If a match is found, packet is forwarded on the link according to the rule
In this case a match is found and the forwarding link is the first link
Each successfully matched packet is also inspected by VTTIF to determine the local traffic matrix
Local traffic matrix as inferred by VTTIF in VNET
Periodically sent to the VNET on the Proxy to form the global traffic matrix
Ethernet Packet is Matched against the Forwarding Table on that VNET
In this case a match has been found and the forwarding link is the destination interface
Hence the Ethernet packet will be injected into that interface

Figure 5. A VNET link.

the virtual machine monitor into the VNET overlay network system. VNET forwards the packet on an overlay link, which may either be realized over the commodity Internet, or through a network that supports reservations (e.g., OMNInet). VNET also supplies the packet to our inference system, VTTIF, for inspection. Local VTTIF agents collect data on each host and regularly aggregate the information on each remote VTTIF instance. A lead VTTIF constructs an estimate of the global application topology among its VMs and the corresponding traffic load matrix. This is passed to the adaptation system, VADAPT.

VADAPT attempts to improve application performance using a variety of adaptation mechanisms. One mechanism is to create new overlay links and corresponding overlay forwarding rules. After VADAPT has chosen a set of new overlay links, it passes it to VRESERVE which creates lightpaths for every link where this is possible. For each new light path thus created, VADAPT then changes the forwarding rules to send the data for the link over the lightpath instead of the commodity Internet.

In the following, we provide more details of the key subsystems, VNET, VTTIF, and VRESERVE.

3.1 VNET

VNET [27, 28] is the part of Virtuoso that creates and maintains the networking illusion that the user's virtual machines (VMs) are on the user's local area network. Each physical machine that can instantiate virtual machines (a Host) runs a single VNET daemon. One machine on the user's network also runs a VNET daemon. This machine is referred to as the Proxy. Each of the VNET daemons is connected by a TCP or a virtual UDP connection (a VNET link) to the VNET daemon running on the Proxy. This is the initial star topology that is always maintained. Additional links and forwarding rules can be added or removed at any time to improve application performance.

The VNET daemon running on a machine opens the machine's virtual (i.e., VMM-provided hook to the VMs' interface) or physical Ethernet interfaces in promiscuous mode. Each packet captured from an interface or received on a link is matched against a forwarding table to determine where to send it, the possible choices being sending it over one of the daemon's outgoing links or writing it out to one of the local interfaces. Figure 5 helps to illustrate the operation of a VNET link. Each successfully matched packet is also passed to VTTIF to determine the local traffic matrix. Each VNET daemon periodically sends its inferred local traffic matrix to the VNET daemon on the Proxy. The Proxy, through its physical interface, provides a network presence for all the VMs on the user's LAN and makes their configuration a responsibility of the user and his site administrator.

3.2 VTTIF

The VTTIF component integrates with VNET to automatically infer the dynamic topology and traffic load of applications running inside the VMs in the Virtuoso system. In our earlier work [12], we demonstrated that it is possible to successfully infer the behavior of a BSP application by observing the low level traffic sent and received by each VM in which it is running. We have also shown [28] how to smooth VTTIF's reactions so that adaptation decisions made on its output cannot lead to oscillation. The reaction time of VTTIF depends on the rate of updates from the individual VNET daemons and on configuration parameters. Beyond this rate, we have designed VTTIF to stop reacting, settling into a topology that is a union of all the topologies that are unfolding in the network.

VTTIF works by examining each Ethernet packet that a VNET daemon receives from a local VM. VNET daemons collectively aggregate this information producing a global traffic matrix for all the VMs in the system. To provide a stable view of dynamic changes, it applies a low pass filter to the updates, aggregating the updates over a sliding window and basing its decisions upon this aggregated view. The application topology is then recovered from this matrix by applying normalization and pruning techniques.

Since the monitoring is done below the VM, it does not depend on the application or the operating system in any manner. VTTIF automatically reacts to interesting changes in traffic patterns and reports them, driving the adaptation process. VTTIF adds little overhead to VNET. Latency is indistinguishable while throughput is affected by $\sim$1%.

3.3 VRESERVE

After VNET has decided which overlay links to create, but before it has created them, VRESERVE analyzes each link to determine if it can be better served using a reservation. Currently this is accomplished through a mapping of default (commodity Internet) interfaces (identified by IP addresses) to interfaces that are connected to a reservable network. If both endpoints of the link share a mapping to the same reservable network, VRESERVE initiates a reservation request for the path between the two corresponding interfaces. If the request succeeds, VADAPT configures the overlay link to use the reserved path. If not successful, the overlay link runs over a path in the commodity Internet.

A key point is that we create an overlay link on top of the reserved path. At first glance this may seem to be redundant, but it allows us to use VNET to perform routing. Without the overlay we would be forced to modify the host machines' routing tables or rewrite the packet headers. With the overlay in place, however, we can perform routing transparently.

Initially, however, this proved to be a substantial performance bottleneck. VNET was designed for the wide area and so did not perform especially well on these very fast links. We redesigned and reimplemented several parts of VNET to improve performance enough to warrant the use of the high speed connection. We discuss VNET and other bottlenecks and our modifications in Section 5.

The actual implementation of VRESERVE is straightforward. It is a Perl module imported by VNET that implements a procedural interface for the creation and destruction of optical lightpaths. VRESERVE also tracks any changes to the reservable network's state made by a caller. Network reservations are made by interfacing directly to ODIN. ODIN consists of a server running on a trusted host and a command-line client. VRESERVE simply constructs and executes command-lines. Because ODIN does not support deferred scheduling VRESERVE immediately indicates success or failure in creating a lightpath.

3.4 Example

A typical execution scenario is as follows. A set of virtual machines V, are started on a distributed set of hosts. A VNET star topology is created, with a proxy machine p, to enable communication for every VM in V. A parallel application is then executed inside each VM in V. All intra-VM communication is routed through p, and a traffic matrix is aggregated by VTTIF. From that matrix VTTIF derives a communication topology amongst the VMs in V. VADAPT uses this topology, combined with the mapping of VMs to hosts, to define a better topology amongst the VNET daemons. This topology consists of a set of overlay links E. We choose k links with the highest bandwidth requirements from E and place them in H, $H \subseteq E$. VADAPT passes H to VRESERVE for action.

VRESERVE analyzes H and determines a subset of overlay links R for which reservations are possible. VRESERVE then requests reservations for each overlay link in R. Links that suffer from path reservation failure are removed from R. VNET then creates the overlay network. This is accomplished by creating an overlay link for each element in H and adjusting the forwarding rules to send packets over the reserved paths for the links in R and over the commodity Internet for $H - R$. As the communication pattern changes, a new set H' is created by VADAPT and passed to VNET. VNET and VRESERVE process all the new links identically as before, generating an overlay network of $H \cup H'$. However following the creation process VNET finds the difference $H - H'$, which corresponds to links not needed in the new topology. It then removes those links, as well as any reservations allocated to links in $H - H'$.

The implementation of VRESERVE is about 400 lines of Perl. Half of this is the VRESERVE module, while the other half interfaces VADAPT to VRESERVE. The majority of the implementation involves parsing the output from ODIN. Modifications to VADAPT take the form of VRESERVE API calls and an added IP address mapping service.

3.5 Limitations and assumptions

In the above and in our experiments, we make the following simplifying assumptions:

- The aim of the application is to increase its throughput.
- All routing on the overlay is shortest path first.

Our heuristic leverages the information provided by VTTIF to make the VNET overlay topology conform to the inferred application topology by adding and deleting overlay

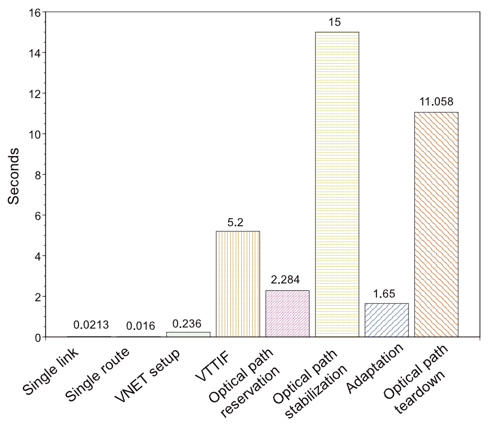


Figure 6. The configuration-time costs for the two VM scenario shown in Figure 1.

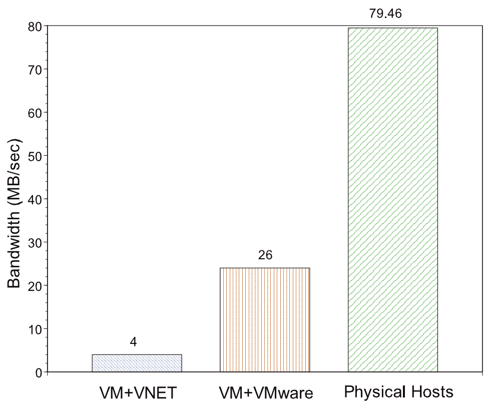


Figure 7. Throughput achieved by ttcp on an optical network.

links and forwarding rules and by using network reservation, where possible.

However, in a complete Virtuoso-like system, the control algorithm of VADAPT must also take into consideration the measured network data. Further, VM migration, CPU reservation, network reservation, overlay topology configuration, selection of overlay link type, and routing on the overlay are simultaneously available as adaptation and reservation mechanisms to improve application performance and avoid annoying security policies. This is a challenging optimization problem which we have not yet solved.

4 Experiments

To evaluate our system we ran several experiments to determine its performance. In the following, we first examine the configuration time of our system, and then evaluate the performance of data transfers through the whole system and the performance of a simple parallel application benchmark. The result is an existence proof: the system works and there is at least one case where it can lead to enhanced performance.[2]

4.1 Configuration time

Figure 6 shows the costs involved in configuring the network using our system. The primary cost was the time spent in the reservation system itself. Creating a path entails two delays. The first is a software delay. It took $\sim$2.5 seconds for ODIN to send the configuration commands to all the switches. The second delay ($\sim$15 seconds) is the time needed for the hardware to reconfigure itself and for the path to stabilize. This stabilization delay is constant, regardless of the complexity of the number of switches being configured. The software delay, however, will grow linearly with the number of switches in a path because ODIN does not currently support parallel configuration. The time taken to tear down an optical path was $\sim$12 seconds.

VTTIF is a significant, but smaller contributor to the setup delay. It must observe traffic for some period of time before it can report a topology. The VTTIF time is a function of a number of parameters and can be as low as one second. For more information, see our previous work [28]. The time to execute VRESERVE and to create the individual overlay links is lost in the noise.

The total time from the start of network communication to channeling packets over an overlay link running through a lightpath is < 30 seconds.

4.2 VM-to-VM TCP performance

In this experiment, we use the configuration of Figure 1 and run the ttcp TCP benchmarking tool in the two VMs. The system notices the sudden communication between the VMs and establishes a lightpath between their hosts. We would expect a dramatic increase of performance thereafter. The physical network is no longer a bottleneck for the system; it is VNET and VMware that become the bottlenecks. Figure 7 shows the results, comparing the raw throughput between the two hosts with the VMware throughput and the throughput using VNET. While acceptable for communication in the wide area (VNET's original design goal), a $\sim$5 MB/s ceiling is far too low. In Section 5, we describe our VNET enhancements to raise this ceiling. Note that VMware is the next substantial bottleneck after VNET. We

[2]Our experiments were limited in scope due to a surprise shutdown of the OMNInet network. Furthermore, during the interval in which we were able to use it, only one path could be reserved.

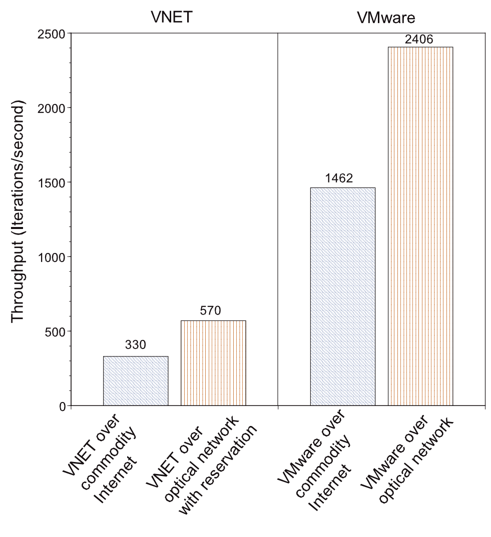


Figure 8. Increasing the performance of a BSP benchmark with an all-to-all communication pattern.

are working to ameliorate this bottleneck as well.

4.3 VM-based BSP benchmark

Here, we use the configuration of Figure 1 and run a PVM bulk synchronous parallel [11] synthetic benchmark in the VMs. The benchmark uses an all-to-all communication pattern, and we measure its execution rate in iterations/second. Figure 8 shows the performance improvement from using a reserved optical network. Performance increases by 170% when the optical path is used. We also see the VNET bottleneck.

The upshot of this benchmark and the preceding ttcp benchmark is that our system can automatically use reservation systems to improve the performance of distributed and parallel applications.

5 Making VNET faster

With very high bandwidth networks, VNET is the bottleneck. VNET was originally designed for predominantly wide-area use. The optical networking scenario has motivated us to dramatically increase its performance through a variety of techniques.

Unfortunately, the OMNInet network became unavailable before we were able to test the improved VNET on it. In the following, our evaluations are on a switched gigabit Ethernet environment.

5.1 UDP overlay links

In the version of VNET used in Section 4, all VNET overlay links are TCP connections. The primary reason was to make it straightforward to support optional SSL encryption. This is not essential, since a VNET link is a virtual Ethernet layer link and thus needs to provide no guarantees of delivery, ordering, or corruption.

Running VNET over TCP results in lower than necessary throughput for applications running inside the VMs. Interaction between the TCP connection at the application layer and the TCP connection used for the VNET overlay dramatically reduces performance. A packet loss in the underlying VNET TCP connection will lead to a retransmission and hence a delay for the application's TCP connection, which in turn could time out and retransmit itself. The application's TCP connection will always detect a packet loss by the expiration of the retransmission timer rather than by receipt of triple duplicate acknowledgments. This will then always trigger slow start instead of fast retransmit, leading to reduced throughput.

VNET now supports creating overlay links using UDP in addition to TCP. This increases the throughput seen by application-level TCP by a factor of two.

5.2 Improved forwarding rule lookup

VNET forwards Ethernet packets. When a packet arrives, it must look up the appropriate forwarding rule based on the packet's destination address. We have improved the lookup mechanism through a forwarding rule cache that gives us constant time lookups on average. This improved performance by a factor of three.

5.3 Where things stand

We have improved the performance of VNET, as measured on a dedicated gigabit Ethernet network, by a factor of six. However, there is still considerable room for improvement and we are a long way from being able to support gigabit speeds in VNET. Figure 9 illustrates where we stand.

The following extensions are planned to improve performance further:

- Enhanced packet filter and packet injection tools. We plan to use the memory-mapped I/O support in pcap to deliver more data from the VM to VNET for each context switch.

- Migration of forwarding functionality to kernel. We plan to move the forwarding core of VNET into the Linux kernel on the host to avoid context switches in their entirety.

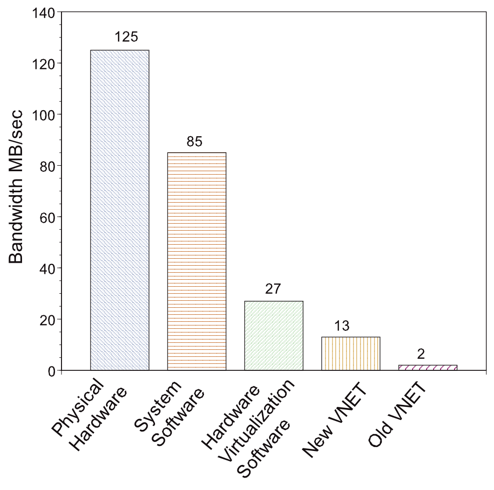


Figure 9. Throughput on a gigabit switch with different bottlenecks.

- Specialized drivers for the VM. We plan to write a device driver for use inside the VM that will more efficiently deliver data to VNET.

6 Related Work

Much work has been done on simulating distributed applications and their communication behavior. Tools such as GridSim [26], SimGrid [2], and Prophesy [30] were developed by the grid community to model an application's behavior with the goal of understanding its computational and communication requirements. Using these models network reservations can be made before the application starts, using simulation results as predictors for network traffic requirements. Our system provides a true *run-time* reservation service that does not require any application simulations. Our system also alleviates the requirement that the user explicitly request advance reservations on behalf of the application.

Run-time adaptation of optical networks to ISP level traffic has been previously demonstrated [10]. Our work takes place at the opposite end of the spectrum; we measure and adapt for individual applications. The other work also treats the optical network as a closed topology, most often seen in the backbone infrastructure of large ISPs. The network topology was modified to reach an optimized state by measuring flow characteristics over the entire ISP. Our project complements this work because we simply make reservations on an optical network and do not care about the physical topology, so long as our bandwidth and latency requirements are met, while their work demonstrates a method of optimizing the physical topology to better meet collective demands.

Advance reservations [25] can be incorporated into optical and other kinds of networks to enhance application and network performance. VRESERVE can easily coexist with advance reservations because on-demand reservation requests are a special case of advance reservations [7]. An early version of VRESERVE specifically targeted operation with deferred reservation requests. While VRESERVE is able to accept deferred reservations, it is unable to make advance reservation decisions as it would have to predict, not just measure, application demands, a service that we have not yet developed

To the best of our knowledge no previous work exists that demonstrates on-demand run-time reservations for unmodified applications. Our system alleviates the need for application developers and/or users to directly interface with the reservation system. Reservation requests are made at run time and are dependent on the applications current communication requirements.

7 Conclusions

We have demonstrated that it is feasible to automatically create network reservations on behalf of unmodified applications, and that such reservations can improve application performance. Specifically, we reserved lightpaths on behalf of applications running in virtual machines by observing their communication traffic over an overlay network and reconstructing their topology from that low-level traffic. Our techniques require no modifications of the application or help from the user or developer.

One question is to what extent our results can generalize. Must we use VMs and an overlay network? Can we support other network reservation models? The answer to the latter question is clearly yes, as the ODIN provisioning model is not qualitatively much different from other per-flow reservation models. We believe that our work can generalize beyond VMs and overlays. For example, traffic monitoring and inference could be done in the kernel and then used to adjust routing tables.

The assumptions in Section 3.5 need to be relaxed to lead to a more general adaptive environment that includes VM CPU reservations [16], VM migration, forwarding, and the overlay topology itself. Also, we have not yet incorporated network measurement data such as from a passive system [33, 13]. A key challenge in our future work is to develop a fast algorithm that can take into account all the available information and choose among adaptation mechanisms, including network reservations, to optimize application performance.

We believe there are clear benefits to using configurable and reservable networks in concert with adaptive overlay network technologies. Adaptive overlays let us seamlessly integrate new networking technologies into existing applications and into the commodity Internet.

Because VTTIF can provide a holistic view of the application, an entire topology and traffic matrix at once instead of just a link at a time, it should be possible for an optical network reservation system to exploit this higher level, detailed information to schedule reservations across the whole network collectively, providing together with sophisticated time-driven scheduling of the VMs, global communication and computation context switches [5].

References

[1] CANARIE INC. Lighting the Path to Innovation, Annual Technical Report, 2003 - 2004.

[2] CASANOVA, H., LEGRAND, A., AND MARCHAL, L. Scheduling distributed applications: the SimGrid simulation framework. In *Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)* (May 2003).

[3] CHAN, V. W. S., HALL, K. L., MODIANO, E., AND RAUSCHENBACH, K. A. Architectures and technologies for high-speed optical data networks. *Journal of Lightwave Technology 16*, 12 (December 1998), 2146–2168.

[4] DIKE, J. A user-mode port of the linux kernel. In *Proceedings of the USENIX Annual Linux Showcase and Conference* (Atlanta, GA, October 2000).

[5] FELDMANN, A., STRICKER, T., AND WARFEL, T. Supporting sets of arbitrary connections on iWarp through comminication context switches. In *Proceedings of the Symposium on Parallel Algorithms and Architectures (SPAA)* (1993).

[6] FERRARI, D., BANERJEA, A., AND ZHANG, H. Network support for multimedia - a discussion of the Tenet approach. *Computer Networks and ISDN Systems 26*, 10 (July 1994), 1167–1180.

[7] FIGUEIRA, S., KAUSHIK, N., NAIKSATAM, S., CHIAPPARIC, S. A., AND BHATNAGAR, N. Advanced reservation of lightpaths in optical-network based grids. In *Proceedings of ICST/IEEE Gridnets* (October 2004).

[8] FIGUEIREDO, R., DINDA, P. A., AND FORTES, J. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)* (May 2003).

[9] FOSTER, I., AND KESSELMAN, C., Eds. *The Grid2, Blueprint for a New Computing Infrastructure*, 2nd ed. Morgan Kaufmann, 2004.

[10] GENCATA, A., AND MUKHERJEE, B. Virtual-topology adaptation for WDM mesh networks under dynamic traffic. *IEEE/ACM Trans. Netw. 11*, 2 (2003), 236–247.

[11] GERBESSIOTIS, A. V., AND VALIANT, L. G. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing 22*, 2 (1994), 251–267.

[12] GUPTA, A., AND DINDA, P. A. Inferring the topology and traffic load of parallel programs running in a virtual machine environment. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing (JSPPS 2004* (June 2004).

[13] GUPTA, A., ZANGRILLI, M., SUNDARARAJ, A., DINDA, P. A., AND LOWEKAMP, B. B. Free network measurement for adaptive virtualized distributed computing. In Submission.

[14] HOO, G., JOHNSTON, W., FOSTER, I., AND ROY, A. Qos as middleware: Bandwidth reservation system. In *Proceedings of the 8th IEEE Symposium on High Performance Distributed Computing* (1999), pp. 345–346.

[15] INTERNATIONAL CENTER FOR ADVANCED INTERNET RESEARCH. http://www.icair.org/omninet/.

[16] LIN, B., AND DINDA, P. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In Submission. A version of this paper is available as Technical Report NWU-CS-05-06, Department of Computer Science, Northwestern University.

[17] LINUX VSERVER PROJECT. http://www.linux-vserver.org.

[18] MAMBRETTI, J., WEINBERGER, J., CHEN, J., BACON, E., YEH, F., LILLETHUN, D., GROSSMAN, B., GU, Y., AND MAZZUCO, M. The photonic terastream: Enabling next generation applications through intelligent optical networking at iGRID2002. *Future Generation Computer Systems 19*, 6 (August 2003), 897–908.

[19] NATIONAL LAMBDARAIL. http://www.nlr.net.

[20] NETHERLIGHT. http://www.netherlight.nl.

[21] RAMASWAMI, R. Optical fiber communication: From transmission to networking. *IEEE Communications Magazine 40*, 6 (May 2002), 138–147.

[22] SHOYKHET, A., LANGE, J., AND DINDA, P. Virtuoso: A system for virtual machine marketplaces. Tech. Rep. NWU-CS-04-39, Department of Computer Science, Northwestern University, July 2004.

[23] SIMEONIDOU, D., NEJABATI, R., O'MAHONY, M. J., TZANAKAKI, A., AND TOMKOS, I. An optical network infrastructure suitable for global grid computing. In *Proceedings of TERENA Networking Conference* (2004).

[24] SMARR, L. L., CHIEN, A. A., DEFANTI, T., LEIGH, J., AND PAPADOPOULOS, P. M. The OptIPuter. *Commun. ACM 46*, 11 (2003), 58–67.

[25] SNELL, Q., CLEMENT, M., JACKSON, D., AND GREGORY, C. The performance impact of advance reservation meta-scheduling. In *Proceedings of the 6th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2000)* (2000), pp. 137–153.

[26] SULISTIO, A., PODUVALY, G., BUYYA, R., AND THAM, C.-K. Constructing a grid simulation with differentiated network service using GridSim. Tech. Rep. GRIDS-TR-2004-13, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, December 2004.

[27] SUNDARARAJ, A., AND DINDA, P. Towards virtual networks for virtual machine grid computing. In *Proceedings of the 3rd USENIX Virtual Machine Research And Technology Symposium (VM 2004)* (May 2004).

[28] SUNDARARAJ, A., GUPTA, A., AND DINDA, P. Dynamic topology adaptation of virtual networks of virtual machines. In *Proceedings of the Seventh Workshop on Languages, Compilers and Run-time Support for Scalable Systems (LCR)* (November 2004).

[29] SUNDARARAJ, A., GUPTA, A., AND DINDA, P. Increasing application performance in virtual environments through run-time inference and adaptation. In *Proceedings of the 14th IEEE International Symposium on High-Performance Distributed Computing (HPDC)* (July 2005). In this volume.

[30] TAYLOR, V., WU, X., GEISLER, J., LI, X., LAN, Z., STEVENS, R., HERELD, M., AND JUDSON, I. Prophesy: An infrastructure for analyzing and modeling the performance of parallel and distributed applications. In *Proceedings of the 9th International Symposium on High Performance Distributed Computing (HPDC)* (August 2000).

[31] VMWARE CORPORATION. http://www.vmware.com.

[32] WEI, J. Y. Advances in the management and control of optical internet. *IEEE Journal on Selected Areas in Communications 20*, 4 (May 2002), 768–785.

[33] ZANGRILLI, M., AND LOWEKAMP, B. Using passive traces of application traffic in a network monitoring system. In *of the Thirteenth IEEE International Symposium on High Performance Distributed Computing (HPDC 13)* (June 2004).