NCP: Service Replication in Data Centers through Software Defined Networking

Vijay Mann, Kalapriya Kannan, Anilkumar Vishnoi, and Aakash S. Iyer
IBM Research - India
{vijaymann,kalapriya,avishnoi,aakiyer1}@in.ibm.com

***Abstract*—Enterprise systems often use replication technology to keep multiple instances of applications or data stores in synchronization. Existing data replication techniques are primarily storage based and are usually applied at coarse time granularity. Furthermore, they can not be used to achieve service replication that requires state synchronization in addition to disk synchronization. In this context, we present NCP: a system that uses network based replication to enable service replication in data centers through software defined networking. NCP overcomes challenges associated with network based replication through the use of server virtualization, multicore and software defined networking (SDN) technologies. NCP allows its users to identify flows based on network addresses and ports and to specify a replication target for each such flow. NCP OpenFlow controller, then, automatically determines the ideal switch for replication, install redirection rules as well as special routing rules for replicated packets in the network. NCP middlebox appliance captures the redirected packets, orders them to reconstruct a network session and replays them on to a primary server and a set of replica servers that were chosen as the target of replication. Our experimental evaluation of NCP demonstrates that network based replication can enable scalable service replication in real time with minimal overheads.**

Keywords: Replication, Port Mirroring, SPAN, Data Synchronization

I. Introduction

Enterprise systems often use replication technology to keep multiple instances of applications or data stores (for example, files or databases) in synchronization. This synchronization may be required for fail-over or disaster recovery (DR). Existing data replication techniques are primarily storage based and these are implemented either at the end host or through an appliance. In storage level replication, disk (or storage medium) contents are kept identical through incremental synchronization. However, storage level replication is typically done at a coarse time granularity as real time replication tends to be costly in terms of computing resources. Furthermore, storage level replication cannot help in replication of service instances because it only replicates storage contents but not memory contents (service state). Service replication requires both disk and memory contents to be replicated.

Existing application synchronization techniques, on the other hand, are usually tightly coupled to specific applications and rely on transferring application data stored in memory from one application instance to another. Session replication in cluster based application servers such as Tomcat [1] are good examples of such replication.

To overcome the above problems associated with storage based and application specific replication techniques, we explore network based replication in this paper. The basic idea behind network based replication is that if all changes to a particular host (referred to as primary server or primary) take place through input network traffic in the form of telnet or ssh sessions, or connections to specific applications running on the server, then capturing those network sessions and replaying them to another set of hosts (referred to as replica server or replica) will ensure disk and state (memory) synchronization between the primary and replica. This in turn, can help in realizing service level replication.

While network based replication using host based approaches has been attempted earlier [2] [3],it could not find adoption in the enterprise due to several reasons. First, network based replication assumes that replica instances are identical (in terms of disk contents and state) to begin with. Second, network based replication also assumes that multiple application instances behave in a deterministic manner, i.e., if two application instances are given an identical input, it will result in identical output and state changes. Both these assumptions may not hold true in a lot of cases. Third, network based replication requires large computing power in order to handle multiple application flows without degrading their throughput. Finally, host based approaches often require modification of host stacks across the data center, which is hard to achieve in practice.

We believe that the above challenges can now be mitigated as server virtualization, multicore and software defined networking (SDN) technologies become widespread. For example, techniques such as virtual machine (VM) cloning can ensure that multiple application instances are identical to begin with and will continue to exhibit deterministic behavior. Scalability issues can be handled through large multicore machines, with 32 or 64 processors, and multiple network interface cards (NICs) that are now becoming increasingly common in data centers. Use of SDN technologies such as OpenFlow can enabled replication and redirection of packets in the network and do not require changes to end hosts, which makes it easier to adopt network based replication in a data center environment.

In this context, we present Network Copy (NCP) - a system for network based service replication in data centers. We make the following contributions in this paper:

C1: We present the requirements and compare the various approaches possible for network based service replication.

C2: We present the design and implementation of NCP OpenFlow controller service that automatically determines the"ideal" redirection switch in the network based on the location of the source and the primary destination of a flow as well as the location of a middlebox appliance and the replica servers. The controller service installs redirection rules to redirect packets to a middlebox and special routing rules to ensure that the redirected packets reach the middlebox appliance.

978-3-901882-50-0

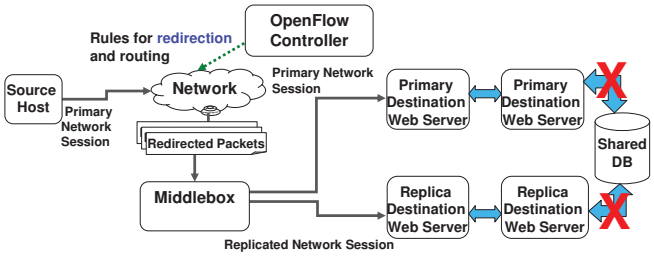

Fig. 1. Source and target environment for network based replication should be isolated from each other and should not have common components

C3: We present the design and implementation of NCP middlebox appliance that captures the redirected packets, orders them to reconstruct a network session and then replays the network session on to a primary destination server and one or more replica servers.

C4: We evaluate the scalability of NCP middlebox appliance using a real testbed with OpenFlow switches and large multicore machines. Our experiments validate our claim that network based replication can enable scalable service replication in real time with less than 5% slowdown of the primary flow.

The rest of this paper is organized as follows. Section II presents the key requirements of a replication service and describes the various possible approaches for network based replication. We present the design and implementation of NCP in Section III. Section IV presents an evaluation of NCP. Section V presents an overview of related research. We summarize our findings and give an overview of our current and future work in Section VI.

II. Methodology

In this section, we first formulate key requirements to enable network based service replication and then describe our technical approach.

A. Requirements

Based on our interactions with various data center system administrators and business unit heads of a large telecom provider in India, we first formulate some key requirements of a replication system.

R1: The replication system should ensure that the performance of the system that is being replicated should not suffer. For network based replication, it implies that network throughput to and from the primary server should not degrade as replication takes place. Administrators were willing to tolerate an increase in replication time if it ensured minimal performance impact on the primary server.

R2: Replication system should ensure that the RTO (Recovery Time Objective) - time taken to recover from a potential outage or to fail-over should not be more than a few seconds. This implies that the replication should be near real time.

R3: The replication systems should be easy to install and configure. It should be largely automated with few manual steps (if any).

R4: Since network based replication to a host can trigger further changes on other hosts, it is critical that the primary environment and the replica environment have no common hosts. If there are common hosts, it will result in multiple copies of the same data being sent to the common host. For example, if the database tier of a 3-tier application is

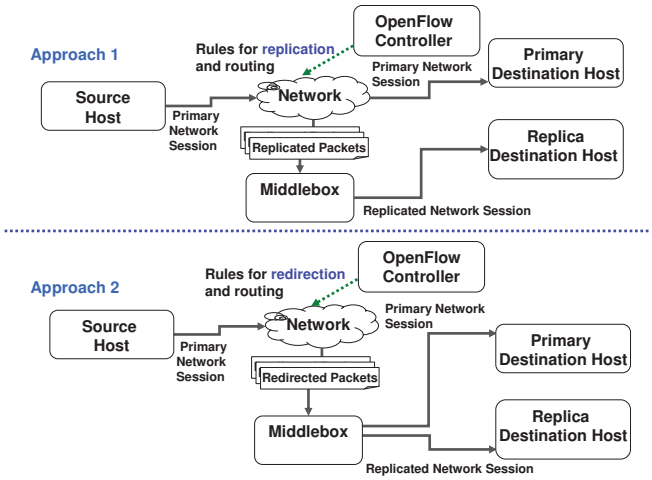

Fig. 2. Different approaches to network based session replication

a common host between primary and replica environment, then it will result in two copies of the same records being generated in the database - once due to the updates sent by the application server in the primary environment, and then again due to the updates sent by the application server in the replica environment. This is shown in Figure 1.

R5: Use of secure protocols such as ssh use public key infrastructure and ssh uses a two-level security protocol in which a session key is negotiated between the end points. Since the primary and replica server typically will have different private keys, the ssh session will have to be be terminated by the replication system, so that data can be first decrypted and then encrypted again before being transferred to replica server.

R6: If primary and replica servers use different credentials, these credentials must be available with the replication service.

B. Approach

In this paper, we discuss network based replication only in the context of connection oriented protocols such as TCP. In this context, network based session replication can be realized through two main approaches (refer Figure 2).

Approach 1: switch based packet replication and middlebox based session replay

Approach 2: switch based packet redirection and middlebox based session replication and replay.

In the first approach, a network switch replicates network packets that match a criteria to a mirror port on the same switch. This is typically referred to as port mirroring or Switched Port Analyzer (SPAN) [4]. However, traditional port mirroring requires that the intended destination of replicated packets be directly connected to the network switch at which replication takes place, as the replicated packets cannot be routed using regular routing protocols (as they have replicated layer-2 and layer-3 addresses). Furthermore, traditional port mirroring does not provide flow level granularity or connection management to maintain state. Traditional port mirroring replicates all incoming or outgoing packets at a particular port. OpenFlow based flow replication can overcome these challenges.

The replicated packets are then routed to a special middlebox appliance which then captures these packets, orders

them and replays the network session on to a replica. In this approach, the only overhead in the primary path is the packet replication at switch. Since this replication is done selectively for user specified flows, this overhead is small and this approach is likely to give good performance. However, this approach requires that the middlebox appliance capture these raw packets using a packet capture library such as libcap [5]. Further, it requires that no packet loss should happen while capturing these packets as there will be no retransmission of replicated packets from the switch. It is well known that most packet capture libraries tend to drop packets at high packet rates [6] and several techniques may be required [7] to overcome this packet loss. Furthermore, a loss of a single packet would mean that the replication process would fail. Due to this reason, we do not follow this approach and do not discuss this approach in the rest of this paper.

In the second approach, a network switch redirects network packets that match a criteria to a special middlebox appliance, which captures these packets, orders them to reconstruct a network session, opens new TCP (or UDP) connections to primary and replica servers, and transfers the captured packets to both primary and replica servers. Note that in this case, the middlebox appliance acts as a proxy to the primary server and lies on the critical path, while in the first case, it does not lie on the critical path between sender and primary server. This approach can slow down the primary flow since the middlebox appliance lies in the critical path. However, as we will demonstrate, a careful design of the middlebox appliance can ensure that this performance degradation is minimal and within the bounds of requirement **R1**.

Note that in both the approaches, the redirected or replicated packets have to be specially routed to middlebox appliance through OpenFlow rules since the destination address of these packets points to the original destination (and not the address of the middlebox). In the first approach, the packets have to be captured through a raw packet capture library, while in the second approach, either a packet capture library can be used or user level sockets can be used if the destination address of the packets can be modified to that of the middlebox (either by the switch itself or at the middlebox appliance through the use of tools such as iptables [8] and ebtables [9]). In the second approach, loss of packets is not a concern, since any such packet loss will be noticed by the source host and it will retransmit lost packets (if TCP is being used).

Both approaches allow an offline asynchronous mode in which the captured network session are persisted to a file and then replayed later. This typically increases the storage requirements of the replication system. While, our system supports this, it should be used only if the RTO defined by requirement **R2** is a few minutes or hours (and not seconds). Use of OpenFlow for packet redirection ensures that we meet the requirement **R3**. We currently do not validate if requirement **R4** is being met and this is left as a sanity check step for the users of the system. Further, we currently do not support requirements **R5** and **R6**, since both require deep packet inspection (DPI) and modifications as well as support for ssh protocol.

III. Architecture and Implementation

We give an overview of our architecure in Figure 3. NCP has two main components - the NCP OpenFlow Controller service and the NCP middlebox appliance.

Flow to be Replicated	Src-IP	Src-Port	Dest-IP	Dest-Port	Middlebox-IP	Replica-IP	Redirection Switch
F1	10.0.0.1	*	10.0.0.8	*	10.0.0.4	10.0.0.6	C

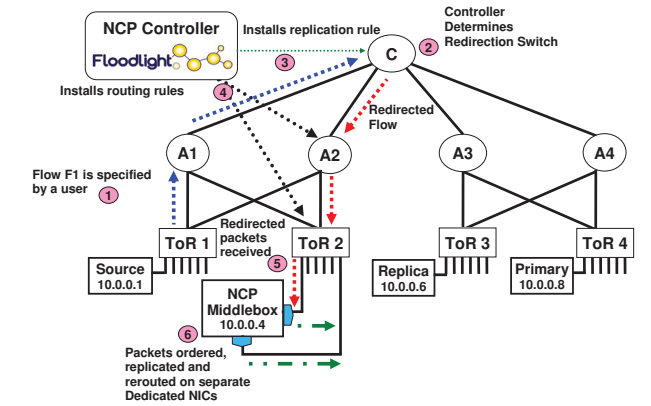

Fig. 3. NCP Architecture

A. NCP OpenFlow Controller

We developed NCP OpenFlow controller service as a module in the FloodLight OpenFlow controller [10]. NCP controller service allows its users to identify flows (based on source and primary destination addresses and ports) and specify a middlebox location and a replication target for each such flow. Once the user saves its preference, the Controller, then, automatically determines the ideal switch (referred to as the 'RedirectionSwitch") in the network that should redirect packets for each flow based on the location of source, primary destination, middlebox and the replication target in the network topology. A simple lowest common ancestor (LCA) algorithm is used to select the "RedirectionSwitch". The "RedirectionSwitch" is the lowest common ancestor of the middlebox and the primary destination. This selection ensures that the packets are redirected at the last possible point in the network. This has at least two benefits. First, redirected packets have to traverse least number of hops to the middlebox. Second, number of special OpenFlow rules required to route the replicated packets are kept to a minimum.

NCP OpenFlow Controller implements the LCA algorithm as follows: it first gets the routes from the source host to primary destination (the primary route) and from the source host to middlebox (the middlebox route) from the routing service module. Once it gets the routes, it traverses the primary route in the reverse direction. Each switch that is encounters on the path is added into a path HashSet. After finishing the primary route, it starts traversing the middlebox box route in the reverse direction. Each switch encountered on the path is added into the same path HashSet. Addition into the path HashSet throws an exception if a switch already exists in the path HashSet. The first time this happens, that switch is the lowest common ancestor between the primary and middlebox and is marked as the RedirectionSwitch.

NCP controller service then installs a redirection rule on the RedirectionSwitch and installs special routing rules on all switches that lie on the path between the RedirectionSwitch and the middlebox. These special OpenFlow routing rules are required since redirected packets have to be specially routed to middlebox appliance through OpenFlow rules as the destination address of these packets points to the original destination (and not to the address of the middlebox). These special routing rules match packets on the input port they arrive

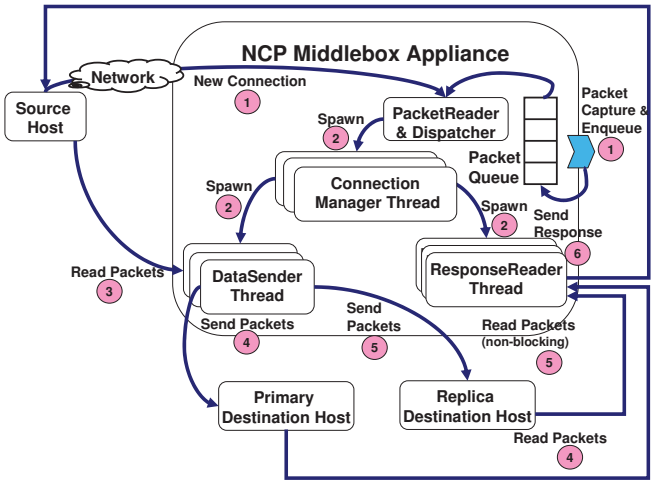

Fig. 4. NCP Middlebox Appliance Architecture

on, the original primary destination address and the source address. The controller can optionally install rules to add an additional header to the replicated packets to tunnel the packets all the way to the middlebox. This additional header comprises of the middlebox address. No additional routing rules to route the redirected packets are required in this case. The additional header is stripped off by the NCP middlebox appliance.

B. NCP Middlebox Appliance

NCP middlebox appliance captures the redirected packets on the middlebox machine either using user level sockets or using a packet capture library such as libcap [5]. User level sockets can be used only if the destination address of the packets received by the middlebox has been modified to that of the middlebox itself - either by the redirection switch or by middlebox appliance kernel through the use of tools such as iptables [8] and ebtables [9].

NCP middlebox orders the captured packets to reconstruct a network session. Similar to packet capture, middlebox has the option to either send raw TCP packets or to send packet payloads over user level TCP sockets. It can chose to either create new TCP (or UDP) connections to replica and primary servers and send the payload (that is, the data portion of the packet minus the headers) of the packets captured or send raw packets that have been captured after modifying the destination and source layer 3 (IP) and layer 2 (Ethernet) addresses appropriately in the packet header. Raw sockets can give a slightly improved performance, but have the downside that TCP's congestion control and retransmission has to be implemented at the middlebox. Creation of new TCP connections to replica and primary servers is preferred because it ensures that packet losses and retransmissions will be handled by the underlying transport layer. NCP middlebox creates new TCP connections to both primary destination and replica servers, and transfers the captured network session to the primary destination as well one or more replica servers.

A schematic of the NCP Middlebox Appliance architecture is shown in Figure 4. NCP Middlebox Appliance has a fully multi-threaded architecture and has been implemented in Java. It uses a combination of jpcap [11] and libpcap [5]. The captured packets are enqueued into a queue (step 1 in Figure 4), which is handled by a PacketReader and Dispatcher Thread. Alternatively, it also supports listening for new connections over user level sockets. For each new connection request (step 1), a new ConnectionManagerThread is spawned which in turn spawns a DataSenderThread and a ResponseProcessorThread (step 2). DataSenderThread reads incoming packets on the given connection from the source host (step 3) and sends the payload for these received packets to the primary destination (step 4) and then to the replica host (step 5). Meanwhile, ResponseProcessorThread continues to read packets in parallel from the primary destination host (a blocking read call in step 4). As soon as it receives some packets from the primary destination host (which could be a result of some packets sent by the DataSenderThread), it attempts to read from the replica host as well (a non-blocking read call in step 5). Note that this read from the replica host is non-blocking (implemented through a socket timeout as well as through input stream API call "available"), since it is possible that replica host may respond late. A non-blocking read call to the replica is essential to ensure that the thread does not stall (in case replica responds in a slightly different manner) and that the primary destination does not get delayed. Data read from the primary destination is sent back to the source host (step 6).

Note that NCP middlebox's multi-threaded architecture allows these threads to be scheduled in parallel on a large multi-core machine and enables it to scale well. Also, note that there are no queues maintained at the middlebox other than a PacketQueue in case of packet capture using libpcap and jpcap. This is because the middlebox does not do much processing of the captured packets. The payload of the captured packets is sent directly to the intended recipients by the different threads. However, as the middlebox supports more functions such as password modifications (as per requirement **R6** from section II) and support for encryption protocols such as SSH (requirement **R5** from section II), appropriate queuing mechanisms will have to be added to handle processing latencies.

IV. Experimental Evaluation

In this section, we present an experimental evaluation of the NCP prototype. Figure 5 shows our experimental testbed. It comprises of 4 large multicore servers. Each server has 12 GB RAM and 2 Intel Xeon CPU E5649 processors (at 2.53 GHz) with 6 cores each (a total of 12 cores on each server). Each server runs Ubuntu 12.04 LTS operating system and has two network interface cards (NICs) attached to it. One network interface card is connected to a 1 Gbps standard ethernet switch (does not support OpenFlow). This is shown as the blue network in Figure 5 and it carries the primary (production) traffic. The 2nd network interface card is connected to an OpenFlow switch (IBM BNT G8264) at 10 Gbps (shown as the black network in Figure 5). The black network carries replication traffic. One of the servers runs NCP middlebox code and performs replication between the source host and the replica host. The OpenFlow switches connect to NCP controller running on a separate server.

We conducted 3 sets of experiments to evaluate the NCP middlebox performance. In each experiment, a set of network flows were generated between the source host and the primary host. NCP OpenFlow controller was used to configure flow redirection and the replication destination was set to the NCP middlebox address. NCP middlebox captured the redirected packets, ordered them to recreate network sessions which were then replayed to the replica server. We initially used the black network for both the primary flows and the replicated flows. This was required since the ability to redirect flows through

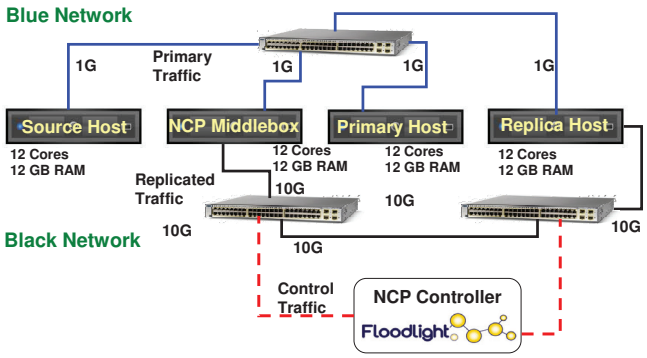

Fig. 5. Experimental Testbed: 4 servers with 12 cores each, and 12 GB RAM: each server has two NICS

an OpenFlow rule exists only in the black network. However, this resulted in a network bottleneck as the same NIC on the NCP middlebox was being used to transfer the primary flows and the replicated flows. These experiments are marked with "shared NIC" in the following results. In order to overcome this, we used the blue network for primary flows and the black network for replicated flows. These experiments are marked with "dedicated NIC" in the following results. Since blue network does not allow redirection of flows through OpenFlow rules, we had to specify NCP middlebox as the destination of each primary flow. The experimental results, therefore, do not capture the overhead incurred due to redirection of packets in the switch and only evaluate NCP middlebox performance. Since, this redirection takes place in switch hardware, we believe that this overhead is minimal and is not likely to change our results in any significant manner.

We experimented with 4 different configurations.

Direct: In this mode, the client connected to the server directly and no replication took place.

NCP - no replication: In this mode, the client connected to the server through the NCP middlebox. However, no replication was performed by the NCP middlebox and NCP middlebox transferred the incoming traffic to the server, acting as a transparent proxy.

NCP - with replication (shared NIC): In this mode, the client connected to the server through the NCP middlebox. NCP middlebox replicated the incoming traffic, and sent it both to the primary and the replica host. In this mode, it used the same NIC (connected to the black network) for both replication traffic and the primary traffic.

NCP - with replication (dedicated NIC): In this mode, the client connected to the server through the NCP middlebox. NCP middlebox replicated the incoming traffic, and send it both to the primary and the replica host. However, in this mode, it used different NICs - NIC connected to blue network was used for sending primary traffic to primary host and the NIC connected to black network was used for sending replication traffic to replica host.

Experiment 1: Iperf Throughput Experiments

In these tests, the primary host ran an Iperf [12] TCP server and the source host executed the Iperf client. Iperf measures the available bandwidth between a server and a client. Number of parallel connections between the client and the server were increased from 1 to 64 using the "-P" parameter. Results are shown in Figure 6 for Iperf throughput between the source host

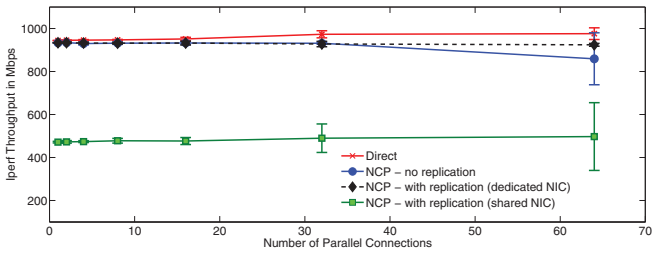

Fig. 6. Iperf throughput experiment: NCP replication performance with a dedicated NIC is at par with a direct connection with a percentage slowdown of 1-2% at 1-16 parallel connections and 4-5% at 32-64 parallel connections.

and the primary host. NCP with replication on a dedicated NIC performs almost equivalent to a direct connection. The primary flows suffer a minor percentage slowdown of 1-2% at 1-16 parallel connections and 4-5% at 32-64 parallel connections. Minimal slowdown in the primary flow was our most important requirement as discussed in Section II and our results demonstrate that NCP satisfies this requirement. Note that NCP also ensures minimal variance (indicated by the error bars). As expected, replication using shared NIC degrades the primary flow throughput to half as the same NIC gets shared between the replicated and primary network traffic. Replicated Iperf traffic (not shown in Figure) reported a throughput of around 450 Mbps across all runs.

Experiment 2: Httperf Experiments

Httperf [13] measures web server performance and outputs both latency and throughput numbers. In these tests, the primary host ran apache web server [14] and the source host executed the httperf client driver [13]. We created a synthetic workload comprising of 5 different file sizes (1KB, 10KB, 30KB, 50KB and 70 KB) which were served by the web server. Httperf client was configured to issue 5 requests for each file size before moving on to the next file using the "wset" parameter (–wset=5,0.2) We used a connection creation rate of 20 requests/second (–rate=20) and varied the number of connections from 100 to 500 (using the "–num-cons" parameter). Each connection was used to send exactly one request, after which it was closed and a new connection was created. A test stops as soon as the total connections specified by "–num-cons" is reached. Results are shown in Figure 7 for average connection lifetime. Average connection lifetime represents the average connection duration for successful connections. This is the time between connection initiation and the time the connection is closed. A connection is considered successful if it had at least one call that completed successfully. Connection lifetime is a direct measure of latency. Even with a mix of workloads, the connection lifetime for NCP replication (with dedicated NIC) is around 10-20% higher than a direct connection as NCP middlebox results in an extra hop that adds to latency. While, this is within the tolerance range for most applications, some extremely latency sensitive applications may get affected. This extra latency can be reduced through various OS/middleware tuning techniques, in-kernel implementation and use of specialized hardware.

Experiment 3: RUBiS Experiments

For our last set of experiments, we deployed a 3 tier J2EE benchmark application called RUBiS [15]. RUBiS is an auction site prototype modeled after eBay.com that is used to evaluate application servers performance scalability. It has a 3 tier architecture with a client driver, an application server (we used JONAS [16]), and a database (we used MySQL [17]).

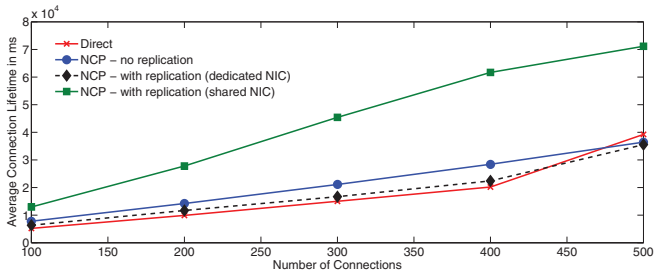

Fig. 7. Httperf latency experiment: Average connection lifetime is around 10-20% higher for NCP replication (with a dedicated NIC) than direct connection.

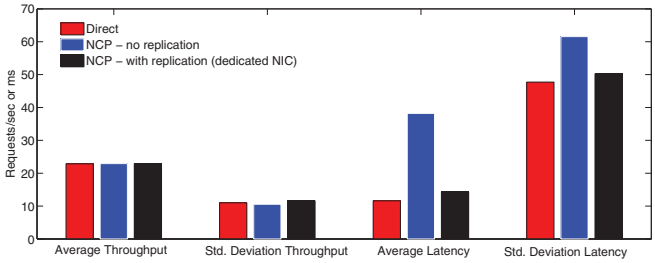

Fig. 8. RUBiS experiments: NCP with replication on a dedicated NIC performs with minimal degradation and has performance at par with a direct connection

RUBiS defines 26 interactions that can be performed such as browsing items by category or region, bidding, buying or selling items, leaving comments on other users and consulting ones own user page. We used the bidding mix workload for our runs that includes 15% read-write interactions, while the rest are read-only browse interactions.

In these experiments, we ran the application server (JONAS) on the source host (refer Figure 5) and two identical database instances on the primary and replica hosts. RUBiS client driver was executed on a different system in the blue network (not shown in Figure 5).Client driver was executed with a workload of 200 clients and with a warmup period of 2 minutes, session time of 5 minutes and a cool down period of 1 minute. We conducted experiments in 3 configurations (we did not do the shared NIC experiment since it was clear that it would have performed bad). First configuration is "Direct" mode in which the application server accessed the database directly. The second configuration is "NCP - no replication" mode in which the application server accessed the database through the NCP middlebox appliance, but no replication was performed. The third configuration is "NCP - with replication (dedicated NIC)" mode in which the application server accessed the database through the NCP middlebox appliance and the middlebox replicated the incoming session to both the databases (running on the primary server and the replica server, respectively). Primary traffic was routed on the blue network and the replication traffic was routed on the black network. The results are shown in Figure 8. Average throughput numbers are almost identical for the three configurations. In the latency experiment, NCP - no replication performs the worse (similar to our observation in earlier experiments). This could be because replication slows down the incoming client traffic resulting in better performance. NCP with replication performs at par with direct mode with an average increase of around 10-15% in the primary flow latency.

In summary, our experiments demonstrated that NCP with replication using a dedicated NIC performs at par with a direct mode configuration.

V. Related Research

Related research can be broadly classified under two categories: Network based replication and Storage based replication.

Network based approaches [18] [19] [2] provide replication by replicating TCP connections. In [18], the end hosts perform the functionality of replicating the data received through their incoming TCP connections to remote hosts. In [2] the traffic is trapped and replayed to a secondary server and transparency is maintained through creation of a virtual NIC in the server. A similar solution is attempted in [20] [21] where properties of TCP protocol have been exploited to provide fault tolerance. Most of these approaches are end host based and require changes to end hosts for trapping and replaying messages. Such modification of host stacks across the entire data center is hard to achieve in practice. NCP, on the other hand, performs packet redirection in the network to a Network Middlebox Appliance and thereby ensures that end hosts are not modified.

Storage replication solutions such as Global Mirroring [22], and Riverbed [19] provide storage replication over long distances. Such solutions typically employ proprietary techniques and require expensive hardware to be deployed to achieve synchronous solution. Several highly available data base and storage systems employ specialized drivers to identify writes to the disks and replicate the writes to a remote server. For instance, IBM SVC controllers [23] use a host based approach to synchronize data written to a SAN disk to another SAN disk. A storage driver on the host machine traps all the writes to the storage subsystem and replays them to a secondary subsystem. Work presented in [24] [25] [26] [27] also falls in this category. Paxos [24] employs 3 way replication that exploits properties of data models to achieve replication. [25] exploits knowledge of working set to replicate data across several replicas. Storage based replication cannot help in replication of service instances because it only replicates storage contents but not memory contents (service state). Service replication requires both disk and memory contents to be replicated. NCP achieves service replication through network based replication.

VI. Conclusion and Future Work

In this paper, we presented NCP (Network Copy) - a system that uses network based replication to enable service replication in data centers through software defined networking. NCP allows its users to identify flows based on network addresses and ports and to specify a replication target for each such flow. NCP OpenFlow controller, then, automatically determines the ideal switch for redirection, install redirection rules as well as special routing rules for redirected packets in the network. Redirected packets are captured by the NCP middlebox appliance which orders them to reconstruct a network session and replays them on to a primary server and a set of replica servers that were chosen as the target of replication. Our experimental evaluation of NCP demonstrates that network based replication can enable scalable service replication in real time with minimal overheads.

We are currently working on extending NCP to handle practical data center scenarios such as difference in passwords used across primary and replica servers and support for encryption protocols such as SSH. These new functions will require deep packet inspection and enhanced packet processing in the middlebox appliance. Maintaining the middlebox scalability under such circumstances will be a big challenge.

REFERENCES

[1] "Tomcat Session Replication." [Online]. Available: http://onjava.com/onjava/2004/11/24/replication1.html

[2] R. R. Koch, S. Hortikar, L. E. Moser, and P. M. Melliar-smith, "Transparent tcp connection failover," in *International Conference on Dependable Systems and Networks (DSN*, 2003, pp. 383–392.

[3] E. Cecchet, G. Candea, and A. Ailamaki, "Middleware-based database replication: the gaps between theory and practice," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '08, 2008.

[4] "Switched port analyzer (span)." [Online]. Available: http://www.cisco.com/en/US/tech/tk389/tk816/tk834/tsd_technology_support_sub-protocol_home.html

[5] "Tcpdump and libpcap," http://www.tcpdump.org/.

[6] L. Deri, "Improving Passive Packet Capture: Beyond Device Polling,," in *4th International System Administration and Network Engineering Conference (SANE)*, 2004.

[7] L. Deri, "nCap: wire-speed packet capture and transmission," in *Workshop on End-to-End Monitoring Techniques and Services (E2EMon)*, 2005.

[8] "iptables - linux man page," http://linux.die.net/man/8/iptables.

[9] "ebtables - linux man page," http://linux.die.net/man/8/ebtables.

[10] "Floodlight: A Java-based OpenFlow Controller." [Online]. Available: http://floodlight.openflowhub.org/

[11] "Jpcap - a java library for capturing and sending network packets." [Online]. Available: http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/

[12] "Iperf Bandwidth Measurement Tool," http://sourceforge.net/projects/iperf/.

[13] "Httperf Web Server Performance Tool," http://sourceforge.net/projects/httperf/.

[14] "Http server," httpd.apache.org/.

[15] "Rubis benchmark," http://rubis.ow2.org/.

[16] "Jonas application server." [Online]. Available: http://jonas.ow2.org/xwiki/bin/view/Main/WebHome

[17] "Mysql database," http://www.mysql.com.

[18] M. Marwah and S. Mishra, "Tcp server fault tolerance using connection migration to a backup server," in *In Proc. Intl. Conference on Dependable Systems and Networks (DSN*, 2003.

[19] Riverbed, http://www.riverbed.com/assets/media/documents/briefs/SolutionBrief-Riverbed-Backup-Replication.pdf.

[20] D. Zagorodnov, K. Marzullo, L. Alvisi, and T. C. Bressoud, "Practical and low-overhead masking of failures of tcp-based servers," *ACM Transactions on Computer Systems*, vol. 27, no. 2, pp. 4:1–4:39, may 2009.

[21] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode, "Migratory tcp: Highly available internet services using connection migration," in *ICDCS*, 2002.

[22] "Ibm global mirror." [Online]. Available: http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100642

[23] "Ibm storage volume controller." [Online]. Available: http://www-03.ibm.com/systems/storage/software/virtualization/svc/index.html

[24] J. Rao, E. J. Shekita, and S. Tata, "Using paxos to build a scalable, consistent, and highly available datastore," *Proc. VLDB Endow.*, no. 4, pp. 243–254, jan 2011.

[25] S. Elnikety, S. Dropsho, and W. Zwaenepoel, "Tashkent+: memory-aware load balancing and update filtering in replicated databases," in *Eurosys*, 2007.

[26] H. Zou and F. Jahanian, "Real-time primary-backup (rtpb) replication with temporal consistency guarantees," ICDCS, Tech. Rep., 1998.

[27] C. Plattner and G. Alonso, "Ganymed: scalable replication for transactional web applications," in *ACM Middleware*, 2004.