RSIP Alternatives for Cray Compute Node External Network Access

Jeremy Enos (jenos@illinois.edu), Paul Wefel (pwefel@illinois.edu), et al
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
CUG2014, Systems Support Category

***Abstract-* Cray systems utilize an implementation of the Realm-Specific IP framework (RSIP) to provide external network reachability from compute resources on the Gemini HSN. On small scale systems, Cray's implementation of RSIP is a workable solution. In large scale Cray systems, such as Blue Waters, the RSIP implementation is inadequate and imposes limitations that reduce the usefulness of the system. Some RSIP limitations encountered by Blue Waters include port exhaustion when the compute nodes try to acquire software licenses from an external server or when trying to allocate compute nodes as condor resources from an external server. It is expected that when remotely steered, large scale visualization runs are attempted, that the RSIP gateways will be an impediment to success. This paper discusses the different solutions for creating NAT configurations, the benefits and drawbacks of each configuration and how they compare to the stock RSIP implementation as provided by Cray in terms of port scalability, performance, security and resilience.**

Introduction

Cray compute systems utilize a custom implementation of RSIP to provide external network reachability from resources on the Gemini HSN such as compute nodes. The current RSIP implementation provided by Cray is lacking in port scalability, performance, security and resiliency, all desirable traits in any high profile, high performance computing system, plus it introduces system configuration complexity. On small scale systems, Cray's "out of the box" implementation of RSIP is a workable solution but In today's large scale Cray systems, such as NCSA's Blue Waters, the RSIP implementation is inadequate and creates limitations that reduce the usefulness of the system.

During the operation of Blue Waters to date, we have observed two instances where jobs were not able to run because there were insufficient network ports available to the compute nodes for external communications due to the RSIP implementation. The first issue was observed when the compute nodes tried to acquire software licenses from an external server. Many short compiles across many compute nodes exhausted the 23 ports allocated to each node mainly do to the small number of ports per node and the RSIP hold timer for port recycling being too long to keep up with the rate of new port requests. The second instance was encountered when trying to provision compute nodes as HTCondor resources from an external server. The project administrators estimated they would need several thousand ports per compute node for the configuration process to succeed. As researchers start to experiment with direct visualization from Blue Waters, we expect the port limitations and throughput to become an impediment when attempting remotely-steered, large scale visualization runs. Our goal is to minimize the technical limitations that Blue Waters presents to researchers as they try to push the boundaries of computational science. Re-working the RSIP implementation will remove one of those technical limitations.

While investigating the above mentioned problems, we learned several interesting facts about the Cray implementation of RSIP which caused us to wonder if there might be a better way of providing external network reachability to the compute nodes that addresses port scalability, performance, security, and resiliency in that order. With that question in mind we set out to discover the alternative options and weigh the benefits against the drawbacks of each and measure how each option compares to the stock RSIP configuration as provided by Cray. We look at several different NAT configurations that could function as a replacement for RSIP on current Cray systems. We look at various scenarios, some as thought experiments and others as proof of concept implementations on our development system to verify functionality. When evaluating a replacement option, we look at several factors, all based on a worse-case scenario of all compute nodes needing external access at once. So let us start by looking at the current RSIP implementation.

About Realm Specific IP

First described in October 2001, Realm Specific IP (RSIP) was drafted and described in IETF RFCs 3102[ref] (full list) 3105[ref] as an experimental protocol that could function as an alternative to Network Address Translation (NAT) and Port Address Translation (PAT). Where NAT and PAT rely on application layer gateways (ALGs) that rewrite portions of the Network Layer and Transport Layer headers, an RSIP gateway instead grants to an RSIP-enabled host access to resources (e.g. Internet Protocol addresses) in another addressing realm. In this way, RSIP allows clients to maintain end-to-end packet integrity and enables applications that would not normally function through a NAT/PAT gateway. RSIP was actually a good choice for Cray to implement and provide with their compute systems. At the time, many applications were not written with NAT in mind and so the applications would embed IP addresses and ports in the data packets effectively breaking the clients connectivity when NAT was in the path. Unfortunately, NAT has become a widely deployed solution within modern networks and as a consequence, there are few, if any, applications that still have issues traversing a NAT gateway. Sadly, this makes NAT now a viable alternative to RSIP and breaks the end to end paradigm of the Internet. However, if handed a technological lemon, make lemonade.

About Network Address Translation

NAT provides a dynamic address translation mechanism where traffic from client machines is rewritten by the NAT gateway to look like it is sourced from a different IP address and port than what it was actually sourced from. The NAT gateway maintains a state table with a mapping of incoming source IP/port to outgoing source IP/port. NAT is limited in sessions by the number of public IP addresses available for translation. Typical deployments use whats called IP overload to map lots of internal clients to a few public IPs. NAT also supports 1:1 assignments which essentially gives each internal client its own external IP. However, it you have enough public IP’s to configure 1:1 NAT you really need to take a look at why NAT is even involved. What about NAT session capacity as a limitation? Today CGN (Carrier grade NAT) is deployed widely and designed to host 100's of thousands of NAT sessions. The University of Illinois uses NAT on its wireless system which regularly sees close to 30k clients active and uses around 240k ports through a single Juniper SRX firewall. Today's hardware is more than capable of handling large NAT demands.

Insert chart showing NAT sessions UIUC wireless

Chart showing NAT sessions limits and bandwidth with iptables, pfSense

RSIP for Cray HSN External Access

The stock RSIP configuration has a permanent kernel module loaded on all the compute nodes and a rsipd daemon running on each RSIP server. In the case of Blue Waters there are 12 RSIP servers active with the clients distributed evenly across the 12 servers. There is a cli command 'xtrsipcfg' available to configure the servers and push the config out to the clients. There isn't anyway to remove the client RSIP module without rebuilding the kernel but you can disable the client association to a server. We hope to be able to run tests with the kernel module completely removed but that depends on Cray building a new kernel and so it will probably be some time before that can happen. So our comparison tests will be run with the client registered and then with the client de-registered from the server. The RSIP client and server communicate periodically with UDP packets to maintain a tunnel state. The default update interval is 1 hour so any changes to the RSIP server config will take up to an hour for the clients to know about it. The update interval is adjustable but there are tradeoffs. At one hour, the client/server RSIP state traffic puts an additional background load on the HSN of an average of eight 70 byte packets / second for an RSIP server with 2240 connected clients. For twelve RSIP servers this comes out to about 100 packets/second of average background traffic so decreasing the update interval by a factor of 2 will double the background traffic to around 200 packets/sec. While these numbers are small, they do represent ‘noise’ on the HSN and so they shouldn't be forgotten.

Another aspect of the stock RSIP configuration is that in our case, each RSIP server came configured to use only its external interface IP for IP/PORT binding assignments and the available port range is defined as 8192-60000. This is what caused our initial application issues with RSIP port exhaustion. With that configuration we have: 12 RSIP servers * 51809 ports / 27064 compute = 23 ports/compute which we discovered becomes quickly exhausted. RSIP has a timeout mechanism for reusing ports which can be made more aggressive but it is not a one size fits all solution and it will only increase the effective port availability slightly. We can increase the number of ports per compute through assigning pools of IP addresses to each RSIP server. When using IP pools, 64511 safe ports are available for each IP (65535-1025). That gives 28 ports/compute/IP address. Using a /24 network for pool assignments across the 12 servers yields

```
 (256 IPs in a /24)
- 2 (network address and broadcast address)
- 1 (router address)
- 12 (addresses for the RSIP servers)
---------------------------------
 241 /12 = 20 IPs per RSIP server
* 28 ports/IP/compute = 560 ports/compute
```

which is a 24x increase over the default configuration. It follows then that using larger IP blocks will provide more ports/compute. So what is the right number of ports/compute? We only have our two cases discussed above to draw from and it would seem that the answer is a couple thousand ports. In a real world use-case it is unlikely that either of the examples would use all 27k compute nodes simultaneously through the RSIP servers. But due to the nature of how RSIP reserves resources and the fact that compute nodes are statically assigned to RSIP servers, each compute is limited to its calculated and configured quantity of ports. Therefore if we have only a few hundred compute nodes that need 10000 ports each, with RSIP we cannot use all the available ports to fill the request. Another limitation of RSIP is that the Cray configuration utility has the ability to configure a failover RSIP server for each compute node to provide resiliency in a failure scenario. However, it turns out the current RSIP implementation doesn't support the failover option so when an RSIP server goes down, due to the static assignments, that's 2240 compute nodes without external reachability until a system administrator brings up a replacement RSIP gateway and updates the clients.

Just NAT

Our first experiment was a thought experiment on the ideal solution which involves numbering all the compute nodes out of public IPv4 space. For multiple reasons, this is typically not a feasible option. Most sites won't have the available IPv4 address space and there is a security concern about making the compute nodes publicly accessible which could be addressed with the addition of a firewall and the tradeoffs associated with firewalls. So the compute nodes end up being numbered out of private IPv4 space and RSIP or NAT is deployed and everyone is happy. But are they really? What would the benefits and drawbacks be with replacing the RSIP servers with NAT servers instead?

If we take the 12 RSIP servers and replace the RSIP process with a NAT process then the compute nodes wouldn't need to run a RSIP module with the overhead associated with it. There wouldn't be the additional background traffic on the HSN from the state packets. The RSIP servers wouldn't need 2240 additional interfaces or need to maintain 2240 tunnels each. Instead the RSIP servers would run a NAT processes, most likely iptables, since the service nodes are restricted to running limux. This allows the additional functionality of firewalling and logging of traffic which will be discussed later. The compute nodes would still be limited by the number of available IPs and ports on each NAT gateway but it wouldn't have to be a statically defined limit and compute nodes could burst to higher numbers of ports if needed. Of course that does leave open the possibility of a small number of compute nodes exhausting all the ports for other compute nodes assigned to that same NAT gateway. We will addresses that concern in a following design. There wouldn't be any resiliency with this kind of NAT solution and it remains to be seen if the throughput would be better or worse than RSIP. Resiliency could be added to this solution but the cost of that resiliency would be needing 2x the number of service nodes in use as NAT gateways. The pairs of NAT gateways would run in an active/passive high availability (HA) mode but that gets tricky as the HSN doesn't support broadcast or multicast traffic which are needed by many HA codes to implement a virtual IP for the NAT pair. Luckily, there appears to be several software packages available that provide a solution to that problem such as linux-ha, heartbeat, and whackamole.

comparison: RSIP vs NAT on service nodes

Ports	RSIP and NAT provide the same qty ports / IP but with NAT compute aren't restricted to a fixed amount per compute
performance	TBD
security	Neither NAT or RSIP are secure. NAT

	implementation through iptables or pfSense gains firewall functionality.
resiliency	RSIP reconfig on failure is subject to keepalive timer of 1 hour and user intervention. NAT reconfig can be done immediately. Can deploy pairs of NAT gateways in HA mode at a cost of additional service nodes to maintain session state.

There are some advantages to a wholesale replacement of RSIP with NAT but can we do better?

Dynamic Routing on Compute Nodes through NAT Gateways

Can we add resiliency to the mix without the 2n hardware penalty? By introducing a dynamic element we could let the compute nodes know which NAT servers are available. One idea is to use routed on the compute nodes and configure the NAT servers to make periodic RIP announcements announcing themselves as a default route. The compute nodes would round-robin through the learned default routes and failed NAT servers would drop out of the route list automatically. Unfortunately there are several issues with this idea with the biggest being that the HSN doesn't support broadcast or multicast announcements which rip relies on for conveying route information. Even if we overcame that hurdle, we have the other issue of non-deterministic loading of the NAT gateways. Also, a single compute could end up sending different flows through different NAT gateways meaning each flow will look like it is coming from a different IP address which would make logging and tracking traffic flows difficult at best. Because the HSN doesn't support RIP updates, this solution is a non-starter but it is getting closer to what we want.

We could alleviate the last two issues by replacing the NAT gateways on the service nodes with simple routing gateways and placing a NAT box on the downstream side of the gateways. The gateways would still announce their availability through rip (if it was supported by the HSN) and the single NAT instance would ensure flows from compute nodes are mapped consistently and that there is a single point of logging for mapping flows to compute.

Redundant *external* NAT Gateways with SDN

The most flexible solution we have come up with so far builds on the previous idea but relies on a form of software defined networking or SDN to handle the compute node to NAT server assignments and gateway failovers. The RSIP gateways are replaced with simple forwarding instances that require minimal customization from the standard service node profile. The addition of some static routes on the compute nodes and gateways is all that is needed. The NAT gateway becomes two external gateways with one operating as a synchronized hot standby triggered by a loss of heartbeat. There is a control process running on the boot node that we will call the keymaster. The keymaster makes the initial compute to gateway assignments at boot time and monitors the gateways health. If a gateway goes down, the keymaster process dynamically changes the route assignments on the affected compute nodes to use a different gateway. If a gateway is overloaded, the keymaster can re-assign compute nodes to other gateways. Since the gateways simply forward packets to the NAT device(s), the gateways can be mapped in and out without affecting traffic flow. With redundant NAT instances, state is maintained for the compute nodes regardless of the gateway it is traversing. Since the NAT gateways reside on external servers, there are many more options available for how it is implemented and instrumented. In our testing of this configuration we are using pfSense, a free open-source BSD based firewall and NAT solution. pfSense not only provides a hot standby NAT implementation but also provides a fully featured statefull firewall with logging, flow data generation and a user friendly web front end for administrating the system. Another viable option would be to use a suite of tools like iptables, conntrackd, and keepalived to create a lightweight redundant NAT system. It should be noted that NAT and RSIP by themselves are not considered a secure solution by the network security community and should not be considered a substitute for a stateful firewall and adequate network filtering. With a topology like this in place, we have the flexibility to add and remove external facing services for the compute nodes. We aren't constrained by the service node architecture or operating system. For example, if a particular job needed to send data to a local pre or post-processor that can't run on Blue Waters, we could programmatically assign the compute nodes involved with the job to use a gateway that has the external processing component networked to it. The following diagram shows what a production deployment would look like.

Insert Diagram of external HA NAT w/ gateways

This solution does have some limitations. One issue could be bandwidth constraints. Based on 1minute samples of network traffic data collected from the RSIP gateways on Blue Water over the past 12 months, any given RSIP gateway never exceeded 70Mb/s tx and 10Mb/s rx . Since all filesystem I/O is handled by a different subsystem, compute node external network traffic is going to be either license server access, visualization, or update packets to an external job monitoring process. There could be other uses but we haven't identified what they are yet. So with current server technology that can drive multiple 40G network interfaces and with the ability to drive 100G interfaces available by the end of 2014 it is safe to say we shouldn't be imposing a bandwidth constraint in exchange for the added benefits of the external NAT SDN architecture. A single off the shelf open-source NAT solution using 40Gb NICs would top out at just under 80Gb/s which would provide enough bandwidth for over 1100 compute nodes at 70Mb/s (4% of BW compute).   In a full scale scenario with all 27064 compute needing simultaneous access, we should theoretically be able to provide close to 3Mb/s / compute. If more bandwidth is needed, the open architecture of this solution allows additional NAT gateway pairs or dedicated hardware solutions to be installed to increase the available bandwidth. The keymaster process can modify the routes on the service node gateways to map in additional NAT gateways. A more economical approach might be to use lower end servers with 10Gb NICs to create NAT pairs for each gateway.

A limitation with the external pfSense 40Gb NAT gateway lies in the return traffic path. Packets traversing the NAT inwards through an established outbound session are currently not load balanced across the gateways. Only one gateway is used per pfSense NAT instance but it does monitor the gateway for failure and a secondary or tertiary gateway is used if the primary fails. This limits return traffic to a single 10Gb/s circuit. The functionality exists in pfSense to load balance return traffic over multiple gateways but at the time of this writing it isn't clear how to implement it. The iptables option doesn't have this limitation as equal cost routes can be used.

Configuration management becomes an issue as the number of NAT gateways increases beyond a single pair. With pfSense each pair of gateways needs to be touched to make any rule or configuration changes. Using iptables with 'firewall builder' allows a central point of configuration for all firewall instances.

comparison: RSIP vs ext HA NAT on service nodes

Ports	RSIP and NAT provide the same qty ports / IP but with NAT compute aren't restricted to a fixed amount per compute
performance	Single NAT HA pair is constrained to max bandwidth through a single server (~80Gb/s). Return traffic limited to 10Gb/s
security	Neither NAT or RSIP are secure. NAT implementation through iptables or pfSense gains firewall functionality.
resiliency	RSIP reconfig on failure is subject to keepalive timer of 1 hour and user intervention. NAT HA happens immediately

	and session state is maintained

iptables NAT Gateways with SDN

In this configuration we are back to utilizing the service nodes as the NAT gateways similar to what was described earlier. The difference with this solution is that we are utilizing SDN via the keymaster process running on the boot node to overcome resource limitations and add resiliency. The keymaster process still assigns compute nodes to NAT gateways but it also reassigns compute nodes based on gateway node availability and load. New NAT gateways can be spun up quickly to provide more resources. Utilizing iptables and conntrackd and firewall builder, a completely instrumented and manageable framework can be built. The only deficiency in this solution is that established tcp sessions will be lost when a NAT gateway fails and the compute nodes are remapped.   That level of resiliency is low on our priority list and not worth the expense of requiring double the service nodes.
The key element of this design is the keymaster process which serves multiple functions. It discovers the number of active NAT gateways at boot time and makes an intelligent assignment of compute nodes to NAT gateways striping the assignments so running jobs are more likely to use a number of NAT gateways instead of all hitting the same gateway. The keymaster process then monitors the gateways reachability during queries to the conntrackd daemon on each gateway. If a gateway isn’t pollable, the compute nodes assigned to it are intelligently remapped to the remaining NAT gateways. The intelligence behind the remapping is derived from the XML session data provided by the contrackd daemon. Doing this will ensure a NAT gateway isn’t overloaded and the compute nodes have access to as many ports as possible.
If a situation arises where there aren’t enough ports available for a particular job and there aren’t any more IP addresses available to assign to the NAT pool then session expiration timers can be adjusted to relieve some pressure on port assignments. If the NAT gateways are saturating their network interfaces then additional service nodes can be mapped in assuming there are available IP addresses for the NAT pool.
Since the NAT gateways are part of the CRAY cluster, the OS and daemon configuration can be managed through normal CRAY processes. Alternately a software package like firewall builder can be used to centrally maintain the NAT and firewall configurations on the NAT gateways.
This iptables NAT gateway with SDN solution is the solution we are planning to test first. Once implemented we will be able to right size the system based on the traffic patterns we observe.

Diagram of this solution

Future Work

Where do we go from here? If Cray were to support numbering the compute nodes with Internet Protocol version 6 (IPv6) we would remove the need for NAT, PAT, and RSIP entirely, enabling an unimpeded, directly routed path between compute nodes and the rest of the

Internet. A stateful firewall can be inserted in the path to alleviate security concerns without impacting the end-to-end paradigm. The compute nodes would speak native IPv6 to other sites enabled for v6 and for those sites that are only v4, the compute traffic can traverse a 6to4 gateway. With IPv6 addressing, having available address space should not be an issue for any site as IPv6 has an adequate supply of addresses and they are easy to acquire. By using IPv6 addresses, as the compute systems scale in size, the addressing can scale accordingly.

Another future enhancement is the implementation of load based monitoring. The mark 1 implementation only allocates gateways based on availability. Using SNMP or a similar polling method we can extract interface utilization from each gateway node and remap the clients to a less utilized gateway. Remapping isn't limited to gateways, using the same SDN process we can also remap clients to other external servers through host routes based on some criterion like load or port usage. The data gathered through this system would be used for capacity planning and identifying hot spots in the exit network design.

Conclusions

What we have presented in this paper is hopefully an open look at what the options are for replacing RSIP with some other network translation technology with the intent of gaining port scalability, performance, security, and resilience. Today, without cluster support for IPv6, the only viable option available is NAT. There are many different ways to implement a NAT solution with the options changing as the requirements change. Weighing all the factors, we favor the iptables NAT gateway on service nodes with SDN approach and will be pursuing a full scale deployment once testing is complete. Due to scheduling issues we were not able to prototype the various options in time for this paper submission but hope to follow up with the results once we have them.

References

- Firewall Builder (http://www.fwbuilder.org/index.shtml)
- Contrackd (http://conntrack-tools.netfilter.org/support.html)
- Wackamole (http://www.backhand.org/wackamole/)
- pfSense (https://www.pfsense.org/)
- iptables (http://www.netfilter.org/projects/iptables/index.html)
- RSIP RFC (https://tools.ietf.org/html/rfc2453)
- NAT RFC (https://tools.ietf.org/html/rfc2663)

--EOF--