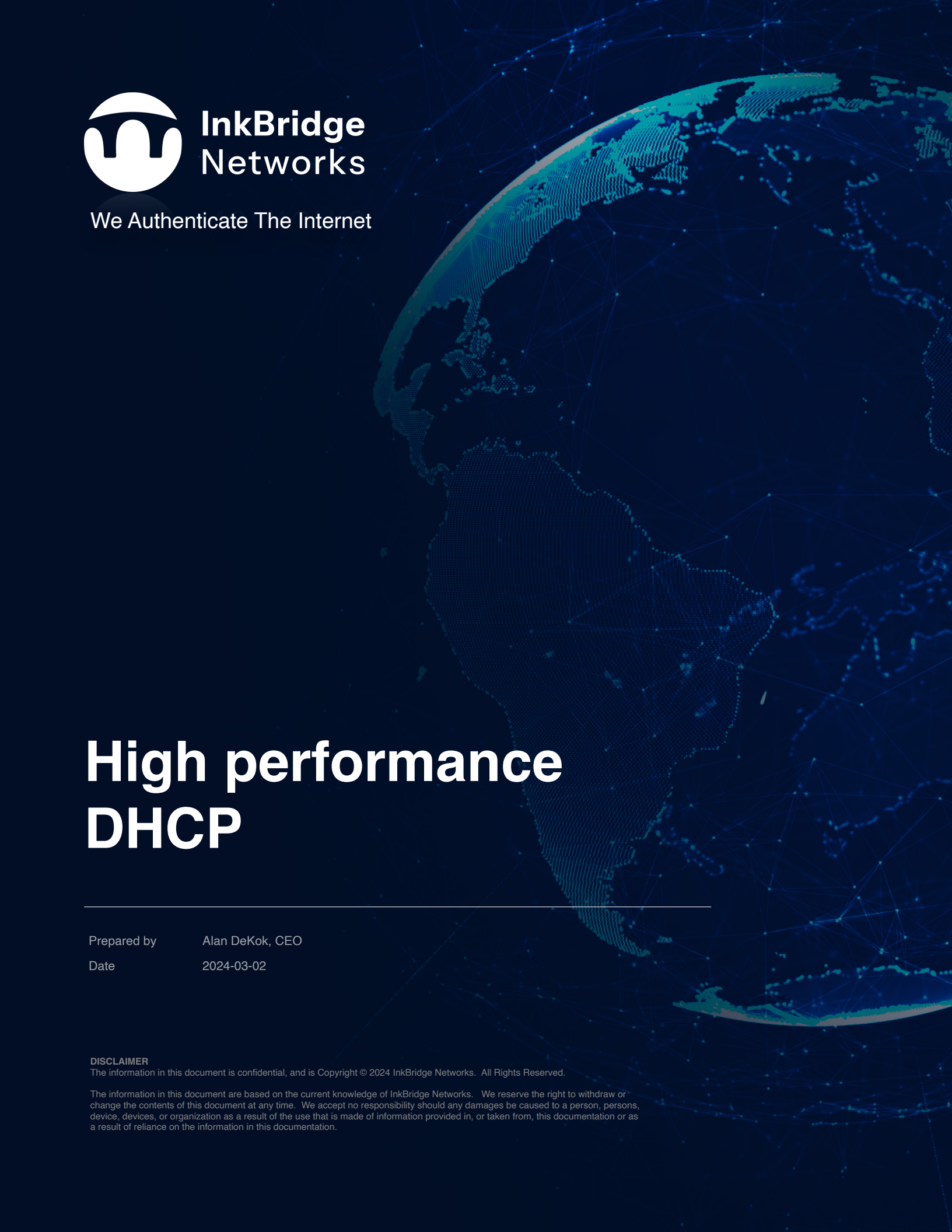

InkBridge Networks

We Authenticate The Internet

High performance DHCP

Prepared by Alan DeKok, CEO

Date 2024-03-02

Executive Summary

This data sheet describes the high performance DHCP server product from InkBridge Networks (formerly Network RADIUS). It allocates 10,000 leases per second per CPU core, on commodity hardware, and using “off the shelf” software. Minimal customization is required, and configuration changes can be made live while the service is running.

For more information, please contact us at: sales@inkbridge.io

We produce the most widely deployed RADIUS server in the world. With more than one hundred thousand (100,000) installations, it has been used in a wider variety of environments than any other product.

Since 2008, we have also produced a DHCP server which is based on FreeRADIUS. It offers unparalleled flexibility, while achieving world-leading performance.

InkBridge Networks

Table of Contents

1. Introduction

DHCP Servers have traditionally offered performance in the range of tens to hundreds of leases per second. Some larger commercial offerings accept higher load, but at a substantially increased cost.

We have built on the proven FreeRADIUS code base. When using Redis Cluster as the back-end database, we achieve unparalleled performance and flexibility. The end result is a system that achieves 10K DORAs (Discover, Offer, Request, Allocate) per second, per CPU core, while offering full security and consistency checks for each lease allocation.

1.1 Why FreeRADIUS

Using a RADIUS server for a DHCP solution may seem to be an unusual choice. For us, the choice comes out of our desire to offer flexible DHCP policies.

Most “open source” DHCP servers offer very little flexibility in the policy rules that they implement. Typically, the leases are assigned from statically configured pools, with a simplistic (and fixed) database back-end. Complex policies are extremely difficult to implement, and integration with multiple databases is impossible.

Commercial DHCP servers are little better. Unfortunately, while they offer simple graphical user interfaces for day-to-day administration, complex tasks and automation must be performed via an extensions API. Additionally, the proprietary databases used in these commercial solutions can become a form of vendor lock-in, making migration to other platforms difficult. This limitation is not acceptable for new deployments.

Adding DHCP support toFreeRADIUS required minimal effort. IP allocation was already available for IP assignment via RADIUS. Supporting DHCP was as simple as adding protocol-specific packet encoders and decoders. The extensible “plug in” nature of FreeRADIUS abstracted away the differences between RADIUS and DHCP, and ensures that neither the internal policy engine, nor the datastore back-ends are aware that they were processing DHCP packets instead of RADIUS packets.

1.2 Performance Optimizations

Some DHCP servers suffer from NIH (Not Invented Here) syndrome. Instead of specializing in the DHCP protocol, they attempt to implement everything themselves, including a simplistic database and API. Their implementations are usually limited to tens or hundreds of leases per second. These limitations arises because the databases fully synchronous, meaning that lease data must be data is written to disk before the lease can be allocated. While safe, this design is slow in practice.

In contrast, FreeRADIUS implements just the DHCP protocol, and relies on external databases to store lease data. This approach allows the administrator to choose the combination of performance vs safety which best meets the needs of the local network.

1.3 Benefits of Redis

Redis cluster as a lease store is particularly interesting, as it has a good balance of scalability, fault tolerance and raw performance. Distributing the operations over multiple nodes in a cluster ensures that the data is copied to multiple slaves, instead of being written synchronously to local disk. The only time that leases can be lost is when the majority of datastore nodes fail at the same time. Outside of a complete power outage with no UPS, this situation is rare.

The Redis IP allocation module developed for FreeRADIUS communicates directly with Redis cluster. It implements the full Redis Cluster protocol, including load-balancing, fail-over and

live node detection. The number of slaves required to report receipt of lease data is tuneable, meaning administrators can make the right trade off between latency and reliability for their environment.

Our tests show this system is capable of allocating 10K leases per second, per CPU core, all on commodity hardware. For redundancy, we recommend using at least two front-end servers, and at least four Redis Cluster nodes. This high availability configuration is capable of handing out 40K leases per second. Configurations with more Redis Cluster nodes can easily reach 100K or more leases per second.

As a high performance solution, it is unparalleled in the cost per unit of performance. The goal for us is to provide the highest performance solution at the most competitive price, and we have achieved that goal. The trade-off is that for now, the product does not have a graphical user interface, and instead relies on direct database modification for configuration changes.

2. Customer Use-Case

We were approached by a customer who operates an ISP in the United States, with a few hundred thousand end users. Their existing DHCP solution was old, and needed replacing. They had approached a few commercial DHCP vendors, and were quoted prices in the range of a few hundred thousand dollars for the system, plus a cost in the high tens of thousands of dollars per year for support. This price was unworkable for them.

The customer was already using FreeRADIUS for the RADIUS side of their business, and were aware that the server included DHCP support. They contacted us to see if we could meet their DHCP needs.

After an investigation, we determined that we could meet their requirements for both total cost of ownership (TCO), and for system performance.

The rest of this document describes how the system was built.

2.1 Databases and Directories Used

We built a system which uses two databases, with each datastore being used where it is most efficient.

Dynamic lease information such as active IP addresses, free IP addresses, and IP / device mappings) were stored in the Redis cluster.

Information which is mostly static was stored in an LDAP directory. This information included subscriber information such as name and MAC address; pool / gateway associations, DHCP options associated with each pool, and static IP addresses.

All information retrieved from LDAP was cached locally. This caching further reduced the load on the LDAP server. It also improved system survivability, which allowed continued operation of DHCP even if FreeRADIUS was isolated from the directory server.

The resulting system has the best of both databases. All IP pool and range information is stored in LDAP, and can be dynamically modified without restarting the DHCP server. All leases are stored in Redis Cluster, which maintains data duplication while offering high performance.

In customer tests of a catastrophic network event, the system successfully brought all end users online within ten (10) seconds. This performance is high enough that network outages due to an overloaded DHCP server are a thing of the past.

The only time that leases can be lost is when the majority of datastore nodes fail at the same time. Outside of a complete power outage with no UPS, this situation is rare.

2.2 System Migration

The customer had an existing DHCP solution, which was in daily use. There was a hard requirement to not have a “flag day”, where the old system went down, and the new system was started. We therefore designed a migration system which required zero down-time.

The first step was to analyze the existing system, and to extract all information about ranges, options, pools, IP addresses, etc. This information was used to build the new DHCP system.

Once the new DHCP system was operational, it was configured to operate on a “span” port. This configuration allowed monitoring of all DHCP traffic. The system was configured to go through the normal DHCP allocation process, but not to reply to the DHCP requests.

Instead, the responses created by the new system were cached, and compared to the responses sent from the existing system. Any differences caused an alert to be logged.

When the two systems had identical responses, we proceeded to the next step. The network was updated so that the new system acted as a gateway,. It received DHCP requests, and relayed them to the old system. Crucially, it also updated the giaddr field so that the responses from the old DHCP server would be sent to the new system.

When it received the response, it would store the response, and update the reply packet with the correct giaddr. The complex policies allowed by FreeRADIUS made this process trivial.

As the clients renewed their leases, the new system would allocate the lease itself, instead of relaying the request to the old system. After a few days of operation, the old system was no longer receiving any packets. It could then be safely decommissioned with no effect on the network.

2.2 Fail-Over and Fail-Back

As with any system migration, there were issues discovered during the migration process. The design of the system ensured that we could easily remove the new system from the network, and fall back to using the old system. Due to the configurable nature of FreeRADIUS, this process could be done live, without affecting service levels.

The ability to migrate the leases without affecting service levels was a critical requirement for the customer, and a large influence in their decision to use FreeRADIUS as their DHCP server.

2.3 How we use Redis

In this section, we describe at a high level how we store DHCP lease data in Redis.

First, the free leases are stored in a Redis ZSET which is ordered by the Unix timestamp. Obtaining a free lease is then an O(log(n)) operation, which scales out to tens of millions of leases on commodity hardware.

Second, the IP addresses are stored in a Redis HASH. The hash contains four keys: range; device; gateway, and counter. The “range” contains the name of the address range to which the IP address belongs. The “device” is a unique identifier for the DHCP client, which is typically the MAC address. The “gateway” is the DHCP gateway through which the allocation request was received. The “counter” indicates how many times the lease has been allocated and released by the client.

Finally, there association between the device and allocated IP address is stored in a Redis STRING type. This information is keyed off of the IP pool and client identifier. This allows the server to detect clients which attempt to allocate an entire range of addresses. When the system determines that a device has an active association, it refuses to allocate new and different IP addresses for that device.

3. DHCP Operation

In this section, we describe how the DHCP service operates. The descriptions are at a high level, and do not include all of the technical details required to implement a full solution.

3.1 DHCP Discover

When a DHCP discover is received, the server looks up the device in the Redis Cluster. If a lease is active for that device, the lease is returned, and processing stops.

An LDAP lookup is performed to discover static IPs. If found, the IP is marked as allocated and is returned to the user.

Otherwise, the Redis ZEVRANGE command is used to find the older expired IP address in the pool. Failure to find a free lease triggers an SNMP trap.

The Redis ZADD command is then used to allocate the lease with a short expiry time. This expiry follows DHCP best practices, and ensures that leases are expired quickly if not acknowledged by the client.

An LDAP lookup is performed in order to return any range-specific DHCP options. This information is also cached in order to minimize the load on the LDAP server.

The Redis hash is then updated with the device information, IP address, expiry time, etc., and the lease is returned to the client.

3.2 DHCP Request

When a DHCP Request is received, the server looks up the device in the Redis Cluster. If no lease is found or the device does not match the allocated lease, a DHCP NAK is immediately returned. This NAK indicates to the client that the lease request was invalid, and that it should start the process again with a DHCP Discover.

Otherwise, the hash is updated with the new expiry time for the lease, and the lease is returned to the client, along with the cached DHCP option information.

3.3 DHCP Release

As with the DHCP Request above, the device is looked up in the Redis Cluster, and a NAK is returned if the device information does not match.

Otherwise, the mapping between IP and device is removed from the Redis Cluster. Dynamic leases are marked as free and returned to the free pool, while static IP leases are simply deleted.

3.4 Lease Management

Lease management is performed by updating the LDAP datastore. DHCP options and static IP assignments can be modified directly in LDAP. Range additions or removals are done via a set of Lua scripts which ensure consistency across the multiple Redis data sets.

At no time does the DHCP server need to be restarted. As the underlying databases are lockless or MVCC compliant, the DHCP server does not even block while datastore operations are being performed.

One key design of this solution is the clear separation between static and dynamic IP addresses. As the databases are not aware of application- specific requirements, they cannot be relied on to perform the “correct” operation when inconsistencies occur. Any conflict between the two is therefore resolved by the DHCP server in its normal operation. This conflict can only be resolved by the DHCP server, as it knows what best to do in order to ensure both network and datastore consistency.

The resulting system is therefore highly flexible, low- cost, and very high performance.

Contact

InkBridge Networks

26 rue Colonel Dumont
38000 Grenoble
France

T +33 4 85 88 22 67
F +33 4 56 80 95 75
W https://inkbridgenetworks.com
E sales@inkbridge.io

InkBridge Networks (Canada)

100 Centrepointe Drive, Suite 200
Ottawa, ON, K2G 6B1
Canada

T +1 613 454 5037
F +1 613 280 1542

InkBridge Networks

We authenticate the Internet