What throughput is really necessary in a PacketCable™ Key Distribution Center?

Abstract

This brief paper discusses the throughput that an MSO should require in a deployed PacketCable Key Distribution Center (KDC). There is a common perception that the usual throughput of a PKINIT-based KDC is too low to be useful; however, by examining realistic deployment scenarios, we show that in fact a typical KDC has more than enough throughput to sustain any reasonable number of deployed subscriber client devices.

Introduction

A typical PacketCable KDC that complies with the PacketCable security specification and running on inexpensive hardware and the Linux operating system can process roughly 10 to 20 incoming AS-REQ messages per second. It is tempting to compare this to the hundreds or even thousands of requests that a high-end provisioning system can service and conclude that CableLabs (which runs the PacketCable project) has somehow completely misdesigned the provisioning process in such a way that the KDC forms an insurmountable bottleneck. In this paper we examine the provisioning process in more detail, to determine whether this conclusion is justified.

Why is KDC throughput so limited?

Most servers nowadays are capable of processing hundreds or even thousands of requests per second. Even enterprise KDCs have similar performance figures. So why is a PacketCable KDC so different? The answer lies in the unique system used by PacketCable to streamline the provisioning process. PacketCable provisioning allows a

client device to be plugged into the system and authenticated without prior detailed knowledge of the device or any manual configuration of the client device by the subscriber. The device authenticates itself to the KDC by using digital certificates that encapsulate public keys. Public key computations are typically very CPU-intensive – this is why, in a system using public key cryptography, one typically wishes to dedicate a machine to the computationally expensive calculations[1]. The heavy load placed on, for example, a hypothetical Call Management Server (CMS) that might be required to perform public key computations (if they did not happen on the KDC) would severely impact the CMS's capability to perform its principal duty of managing calls[2].

A compliant KDC (*i.e.*, a KDC that fully complies with the requirements in the PacketCable security specification) must ensure that incoming sets of certificates chain correctly, which is not an inexpensive computation. The KDC also generates a set of Diffie-Hellman parameters, from which it derives a Diffie-Hellman shared secret, which is in turn used to encrypt a portion of its reply. All of these computations are expensive if performed properly. As a result, a compliant PacketCable KDC is a CPU-bound device; *i.e.*, the power of the CPU determines the throughput.

Shortcuts can be used to increase the apparent throughput – but only at the cost of compliance with the requirements laid out in the PacketCable security specification, and such shortcuts threaten interoperability or -- more likely, more subtly and more seriously -- the very security that the KDC is supposed to enforce. For example, the KDC could save considerable time by not bothering to enforce the correct (and somewhat complex) certificate chaining rules and tests for certificate compliance mandated by the security specification. Or it could use only a small set of pre-calculated Diffie-Hellman parameters to generate a small pool of Diffie-Hellman shared secrets that may be re-used. Using such tricks, one could boost the performance of the KDC many-fold – but it would no longer be a compliant device (although it would inter-operate perfectly), and it would offer only the illusion of compliance and security.

So, a compliant KDC is necessarily going to be "slow" in terms of the number of transactions that it can process per second. We have found that running under a Linux operating system on typical off-the-shelf inexpensive hardware, typical throughput is in the range of ten to twenty PKINIT AS-REQ messages processed per second, per CPU in the computer. At first glance, it seems obvious that this is too few to support any substantial network of subscriber clients. So it is reasonable to ask, what throughput is actually needed to support a large-scale deployment of subscriber clients?

Kerberos and PacketCable Provisioning

In PacketCable, the subscriber clients that reside either in or attached to the home are known as Multimedia Terminal Adapters, or MTAs. The basic process by which they obtain access to network servers such as a PacketCable Provisioning Server or a Call

[1] In the typical "secure" transaction that an ordinary user sees – the interaction between a Web browser client and a Web server using https – the server does not authenticate the client. Hence, the server is not required to perform these calculations.

[2] There are other reasons why an architecture that ties security to a centralized security server such as a KDC is a Good Idea, but they are beyond the scope of this paper.

Management Server is to obtain a "ticket" for said service from the KDC. In a typical example, the flow contains four standard Kerberos messages (we omit some of the details of the contents of these messages for the sake of clarity):

1. The MTA sends an AS-REQ message to the KDC.
2. The KDC authenticates the MTA (using digital certificates embedded in the received AS-REQ, per the PKINIT requirements); it also generates a set of Diffie-Hellman parameters, from which it derives a shared secret which is used to encrypt part of the reply. This reply is known as an AS-REP. Included in the AS-REP is an encrypted "ticket" for the service that the MTA wishes to access.
3. The MTA authenticates the KDC (using a similar procedure that was used by the KDC). It extracts the ticket and uses it to generate a message known as an AP-REQ, which is sent to the server whose service the MTA wishes to access.
4. The server checks that the ticket is valid (although the ticket is encrypted, this is a computationally lightweight calculation because it uses an ordinary symmetric-key algorithm instead of the expensive public-key calculations performed by the KDC). The server returns an AP-REP that is then used by the MTA to create a security association between itself and the server.

We see from this flow that the KDC is involved only in processing the AS-REQ and in generating and sending the AS-REP. We also see that the KDC is the only server that is required to perform the complex public-key calculations. By performing the authentication process and generating a ticket that can be cheaply checked by a server, the KDC is saying to the server: "I have already checked this client's credentials; you don't need to do so".

The important token in this process is the ticket. Tickets are not needed every time that a client wishes to access a server. Hence, the client does not need to contact the KDC each time, for example, that it wishes to place a phone call. Instead, tickets are issued with a lifetime. So long as the ticket has not expired, the client can simply re-use the same ticket whenever necessary, without requesting the KDC to issue a new one. The PacketCable security specification requires that the lifetime of tickets not exceed seven days; in its default configuration, the IPfonix, Inc. KDC will generate tickets whose average lifetime is 6.5 days. This corresponds to a mean ticket lifetime of somewhat more than half a million seconds. In other words, under normal operating circumstances, a single MTA will request a ticket from a KDC once every half a million seconds. Or, to put it another way, a KDC that can service a single request per second could, in theory, support an upper limit of half a million clients.

Needed Throughput

Naturally, there are some other important factors that one must include to determine the actual throughput that is necessary for a particular deployment. We will, for the moment, ignore the fact that concerns related to redundancy and reliability would probably require that more than one KDC be used. Let us consider just a single KDC, even though in practice KDCs are likely to be deployed in groups of two or more.

When the IPfonix, Inc. KDC receives a request for a ticket, it does not automatically grant a ticket for exactly seven days. To do so would naturally lead to a large non-random component in the load on a KDC. Seven days after the initial request was made, the client would request a new ticket. This would result in a heavy daytime load corresponding to the hours in which MTAs were initially installed, and a negligible night-time load, corresponding to the hours in which no MTAs were installed. So the IPfonix, Inc. KDC distributes the lifetime of tickets over a period between six and seven days. This results in a mean lifetime of 6.5 days, which although it is slightly less than the theoretical PacketCable maximum of seven days, means that the requests for new tickets arrive in a much more random fashion over the course of any given 24-hour period.

Tickets whose lifetime varies in this manner result in requests arriving in a manner that correspond to Poisson statistics (we discount new MTAs being booted for the first time, which will still tends to occur during working hours – once there are enough MTAs deployed for throughput issues to rise to the level of being a consideration, such MTAs add only a minor perturbation to the underlying Poisson statistics).

Poisson statistics dictate that the following holds:

$$p(k) = \frac{\exp(-\lambda) \times \lambda^k}{k!}$$

where:

$p(k)$ is the probability of k events occurring in unit time
k is the number of events occurring per unit time
λ is the mean number of events that occurs per unit time

We can use this to estimate directly the necessary throughput of a KDC needed to support a given deployment. Note, however, that we already know the *average sustained throughput* that is necessary: it is simply the number of MTAs divided by the mean lifetime of a ticket.

We can, though, use the Poisson formula to calculate the maximum instantaneous throughput necessary if a KDC is to service every incoming request without ever requiring an MTA to retry. It is important to recognize that this is an extremely conservative estimate, since MTAs are required to follow a well-defined retry sequence if the KDC fails to respond to their first request. In other words, if the KDC is so overloaded that it does not respond to the initial request, the MTA will retry (several times) before giving up. The entire protocol is designed so that the MTA actually asks for a new ticket several minutes before the old one expires. In any period as long as a few minutes, the actual number of requests arriving at the KDC will closely approximate the average sustained rate we describe above. Under normal operation, substantive deviations from this value will occur only over short intervals measures typically measured in seconds or, at most, tens of seconds.

KDCs, like most modern servers, are typically multi-threaded. This means that they can support simultaneous processing of several requests. The result of this is that while a

KDC may have a maximum sustained throughput of *x* requests per second, it may easily be able to accept and process several times *x* requests for short periods of time. We consider this in more detail below.

Examples

It is useful to look at a couple of illustrative examples that illuminate the practical effect of the Poisson equation.

Example 1: 500,000 MTAs deployed

Suppose that an MSO deploys a primary line service with 500,000 MTAs. This means that the mean number of incoming requests per second at the KDC is 500000/561600, or 0.8903 requests per second. This is the average sustained rate at which requests are received. In other words, a KDC that can process requests at the rate of one per second would be adequate to support half a million MTAs (albeit with little room to grow) if those MTAs were to supply uniformly distributed requests.

From the Poisson formula, we can calculate the expected actual number of requests per second that the KDC will receive:

$p(0) = 0.4105$

$p(1) = 0.3654$

$p(2) = 0.1626$

$p(3) = 0.0482$

$p(4) = 0.0107$

This means that in roughly 99 one-second periods out of 100, a KDC will receive three or fewer requests.

In the remaining roughly one-percent of one-second slots, it will receive four or more requests. However, even if the KDC is capable of a sustained throughput of (say) only three requests per second, because of the multi-threaded nature of the KDC, it can still accept all four requests and process them. It can continue to accept requests until all the threads are fully occupied with processing requests. This will happen only if a sudden and sustained burst of traffic arrives at the KDC, such that all threads become occupied before any of them has been able to complete processing.

If the KDC had (say) ten usable threads, it would have to receive the equivalent of four requests per second (three of which it could process fully) for ten successive seconds before its threads were fully occupied and it would begin to receive requests that it could not process. The probability of this happening is so small as to be utterly negligible (approximately 10^{-20}). Therefore, because of the nature of Poisson statistics, a KDC that

can process all received messages in 99% of all one-second intervals is more than adequate to the task.

For the example of 500,000 deployed MTAs, then, a single KDC that processes three requests per second is more than adequate to support the deployed MTAs.

Example 2: 1,500,000 MTAs deployed

Now suppose that an MSO deploys a primary line service with 1,500,000 MTAs, or three times as many MTAs as in Example 1.

Now the mean number of incoming messages per second is 1500000/561600, or 2.6709 requests per second. As before, this is the average sustained rate at which requests are received. It is no surprise that this is exactly three times the average sustained rate derived in Example 1. In other words, a KDC that can process requests at the rate of three per second would be adequate to support one and a half million MTAs (with a little room to grow) if those MTAs were to supply uniformly distributed requests..

From the Poisson formula, we can calculate the expected actual number of requests per second:

p(0) = 0.0692

p(1) = 0.1848

p(2) = 0.2468

p(3) = 0.2197

p(4) = 0.1467

p(5) = 0.0784

p(6) = 0.0349

p(7) = 0.0133

That is, in roughly 99 one-second periods out of 100, a KDC will receive six or fewer requests (note that even though the number of MTAs has tripled as compared to Example 1, and the maximum sustained throughput has also tripled, the instantaneous throughput necessary to handle all requests without dropping any packets has effectively only doubled, from three to six requests per second).

Without going through the same details as were provided in Example 1, we see that for the example of 1,500,000 deployed MTAs, a single KDC that processes six requests per second is more than adequate to support the deployed MTAs.

Power Loss and Restoration

We have considered only the situation in which MTAs are well-behaved and request tickets when they expire, roughly every 6.5 days. There is one additional complication, though. What happens if a substantial number of MTAs lose power and power is restored to them simultaneously?

Primary Line Service

There are a couple of considerations here. The first is that if the MSO has deployed primary line service, then the MSO has effectively promised the subscriber that the phone can still be used even when commercial power is not available. This may be accomplished, for example by providing network power or some scheme involving batteries. The important point, though, is that power to the MTA will not have been cut even though commercial power has been lost. Therefore, to the MTA, the situation is exactly as if power had never been removed, and it will operate on the 6.5-day schedule exactly as in the examples above.

However, if the MSO is providing only secondary line service, then the power will have been removed from the MTA. What happens now when power is re-applied?

Secondary Line Service

MTAs are required to maintain their tickets in nonvolatile Random Access Memory (NV-RAM). Consequently, even if power is lost and then restored, the MTA has access to its tickets without contacting the KDC.

Thus, when MTAs reboot after a power failure, almost all of them will have valid tickets available at reboot time and hence will not access the KDC. Only those MTAs whose tickets chanced to expire during the power outage would need to acquire new tickets. Following our example of tickets that have a lifetime of 6.5 days, this percentage is simply:

(duration of power outage) / 561600 * 100%

For a one-hour outage, this is 0.64% of the MTAs. For a 500,000-MTA secondary-line deployment *all of which lose power for the entire one hour and then have their power restored simultaneously*, this corresponds to 3205 MTAs. Note, however, that all 500,000 MTAs (and their Cable Modems) must pass through all the other provisioning steps (including cable modem ranging and registration). So, in order for the KDC to be the bottleneck even in this extreme example, it must process requests a factor of approximately 200 times slower than any other component in the registration and provisioning system.

Even if the KDC *were* running on a platform that was so slow that it did become the bottleneck, only a tiny fraction of subscribers would be affected (since only 0.64% of them have MTAs that require new tickets). Assuming that the MSO has only a single KDC and the KDC can process only a few requests per second, then at most a few minutes will go by once power is restored before 100% of subscribers will have full

service. And remember that these are, by definition, subscribers to a secondary-line service.

In a large-scale deployment of secondary-line service, it is incumbent on the MSO to decide how much KDC overcapacity he should have available in the head-end. Some of the factors entering into such a decision would likely be the answers to questions such as:

1. How many subscribers do I have?
2. What is the normally needed maximum sustained KDC throughput needed?
3. What is the probability of 1% of my customers losing power and having it restored simultaneously? How about 10% of my customers? 50%? 100%?
4. How long am I willing to wait before 100% of my secondary-line customers are back online.
5. What is the ranging/registration/provisioning throughput of other devices in my network?

The MSO may decide that overprovisioning KDC throughput by a factor of two is adequate, or he may decide that he needs to overprovision by a larger factor. This is a business decision affected by the relative importance the particular MSO places on the answers to questions such as those given above.

We do note in passing that a reasonable basic large deployment would in any case likely have a sustainable KDC throughput of several tens of requests per second (simply because a pair of ordinary uniprocessor Linux KDCs typically provide this sort of throughput), so that even in the catastrophic (and extraordinarily unlikely) condition of half a million subscribers losing power *and then having it restored simultaneously*, the MSO might reasonably decide that the normal basic throughput of his system will allow secondary-line customers to return to the network at an adequate rate[3].

Conclusion

By examining the statistics associated with large-scale MTA deployments, we have concluded that MSOs that provide primary-line service are adequately served by quite minimal throughput on the KDC(s), requiring support for only a few transactions per second.

Deployers of secondary line service require support for only slightly increased throughput to bring all their subscribers back online in a short period following even the most major of outages.

[3] The extremity of this example causes us to wonder when was the last time that several million people (who might conceivably include half a million secondary line subscribers) simultaneously had power restored after an outage. Wide scale power failures are usually due to some kind of disaster, and power is restored piecemeal over the affected area. In fact, we wonder if the underlying electricity system could accommodate such a sudden wide-scale restoration of power as that described in our example.