hFT-FW: hybrid fault-tolerance for cluster-based Stateful Firewalls

P. Neira[1], R.M. Gasca[3]
Department of Languages and Systems
Quivir Research Group
ETS Ingeniería Informatica
University of Sevilla
Avda. Reina Mercedes, s/n
41012 SEVILLA - Spain
{pneira|gasca}@us.es

L. Lefèvre[2]
INRIA RESO - Universite of Lyon
LIP Laboratory
(UMR CNRS, INRIA, ENS, UCB)
Ecole Normale Supérieure de Lyon
46 allée d'Italie
69364 LYON 07 - France
laurent.lefevre@inria.fr

***Abstract*—Failures are a permanent menace for the availability of Internet services. During the last decades, numerous fault-tolerant approaches have been proposed for the wide spectrum of Internet services, including stateful firewalls. Most of these solutions adopt reactive approaches to mask failures by replicating state-changes between replicas. However, reactive replication is a resource consuming task that reduces scalability and performance: the amount of computational and bandwidth resources to propagate state-changes among replicas might be high. On the other hand, more and more commercial off-the-shelf platforms provide integrated hardware error-detection facilities. As a result, some current fault-tolerance research works aim to replace the reactive fault-handling with proactive fault-avoidance. However, pure proactive approaches are risky and they currently face serious limitations. In this work, we propose a hybrid proactive and reactive model that exploits the stateful firewall semantics to increase the overall performance of cluster-based fault-tolerant stateful firewalls. The proposed solution reduces the amount of resources involved in the reactive state-replication by means of bayesian techniques to perform lazy replication while, at the same time, benefits from proactive fault-tolerance. Preliminary experimental results are also provided.**

I. Introduction

Failures are a permanent threat for continued availability of the Internet services. If failures are not handled appropriately, they can lead to service misbehaviours and disruptions.

During the last three decades, numerous fault-tolerant approaches have been proposed for the wide spectrum of Internet services. This includes databases [1][2], web servers [3], TCP-based back-end servers in general [4][5][6], VoIP PBX [7], stateful firewalls [8][9], CORBA [10], among many others. These solutions are mainly based on active and passive replication. Thus, they inherently adopt reactive approaches to mask failures.

However, replication is a resource consuming task. The amount of computational and bandwidth resources to propagate state-changes among replicas is generally high. Thus, reactive fault-tolerant solutions may not be suitable for large scale and high performance network setups.

On the other hand, commercial off-the-shelf platforms provide integrated hardware error-detection facilities more and more. These mechanisms go even further as they can also correct hardware errors in runtime. These include RAM memory and PCI bus transfer error detection and correction [11].

Intuitively, if a system has correctable errors, the service will experience performance degradation. Moreover, correctable errors may become uncorrectable at some point, and the chances of experiencing a failure increases. Following this basis, it would be safe to proactively migrate the service from the primary replica, that is experiencing correctable errors, to a sane operational backup replica; thus, forcing the take-over.

However, a pure proactive approach is risky because:

1) It lacks of completeness since, as for now, there is no feasible model to diagnose and predict all kind of possible computer software and hardware errors in runtime. At best, they cover a subset of the possible errors.
2) If the failure happens during the service migration, we may fail to recover the service or, even if we try to recover the service partially, we will not be able to know how many states have been recovered either. This is due to the fact that we have no guarantees on when an error turns out uncorrectable and, consequently, leads to failures.

For that reason, we propose a hybrid architecture to solve these issues which is composed of two parts:

1) Reactive fault-handling: the service state-changes are preventively propagated from the primary to the backup replicas. However, we exploit the semantics of Internet service to relax the degree of replication. Thus, reducing the waste of computational and bandwidth resources.
2) Proactive fault-avoidance: the service states are fully migrated from the primary replica to a sane backup replica in case that some errors are detected.

Thus, in contrast to other previous work in this area, we do not aim to replace the reactive fault-tolerance approach with a pure proactive solution but, instead, to hybridize both approaches to:

1) define an architecture more suitable for large scale

environments.
2) reduce the impact of a possible unsuccessful proactive take-over.

However, covering the whole variety of Internet services, with very different semantics, would be rather ambitious. For that reason, we particularly focus on cluster-based fault-tolerant stateful firewalls as case study to extend our previous works in this field.

This paper is organized as follows: In Section II, we provide an overview of related works in the domain of fault-tolerant stateful firewalls and existing proactive fault-tolerant architectures. Then, Section III details the system model covered in this work. The hybrid proactive and reactive architecture is detailed in Sections IV, V and VI. We conclude with the evaluation in Section VII and the conclusions and future works in Section VIII.

II. Related Works

In our previous works, we have proposed an event-driven architecture and a replication protocol to build cluster-based reactive fault-tolerant stateful firewalls [8]. We have extended it to support multi-primary setups in [9]. The main features of the solution are:

1) Transparency. The solution ensures negligible delay in client responses and quick recovery from failures. Clients does not notice any bandwidth throughput drop. It is suitable for 1 GEthernet network setup.
2) Simplicity. We reuse and extend existing software-based, high availability solutions. Moreover, the client does not require any modification. Therefore, this is a client transparent solution. The firewall must also require minimal and non-intrusive modifications.
3) Low cost. The solution is suitable for off-the-shelf equipments and it requires no hardware extensions.
4) Multi-primary workload sharing setups. The architecture proposed supports advanced setups where several replica firewalls share workload to increase scalability.

In particular, the replication protocol exploits the stateful firewall semantics to reduce the number of retransmitted messages under message omission situations and improve the overall flow durability.

On the other hand, several research works have focused on defining frameworks to predict and diagnose failures in computer systems [12]. Indeed, these are the main block to build proactive fault-tolerance solutions for Internet services.

In [13], the authors provide a proactive event-driven fault-tolerant framework based on virtualization techniques that aims to improve the scalability of fault-tolerant High Performance Computing (HPC) solutions. The solution is composed on three blocks: the fault predictor (FP), the policy daemon (PD) and the fault-tolerance daemon (FTD). In this approach, the FP asynchronously delivers alarms to the PD which determines how to react to the detected error according to the selected policy. The policy is expressed in a state-machine specification. The decision issued by the PD is executed by the FTD which migrates the virtual machine to a healthy replica. A similar solution is presented in [14] that directly implements the policy into the system.

In [15], the same authors of [13] make some interesting observations regarding reactive and proactive fault-tolerance. Among them, they state that all system failures cannot be predicted and, consequently, proactive fault-tolerant policies are still very naive since they are still based on basic fault prediction mechanisms.

III. System Model

The formalization of the stateful firewall model is out of the scope of this work as other works have already proposed a model [16]. Nevertheless, we formalize the definitions extracted from the fault-tolerant stateful firewall semantics that are useful for the aim of this work:

Definition 1. Fault-tolerant stateful firewall cluster: it is a set of stateful replica firewalls $FW = \{fw_1, ..., fw_n\}$ where $n \geq 2$ (See Fig. 1). The number of replica firewalls n that compose the cluster depends on the availability requirements of the protected network segments and their services, the cost of adding a replica firewall, and the workload that the firewall cluster has to support. We also assume that failures are independent between them so that adding new replica firewalls improve availability. The set of replica firewalls fw are connected through a dedicated link and they are deployed in the local area network. We may use more than one dedicated link for redundancy purposes. Thus, if one dedicated link fails, we can failover to another.

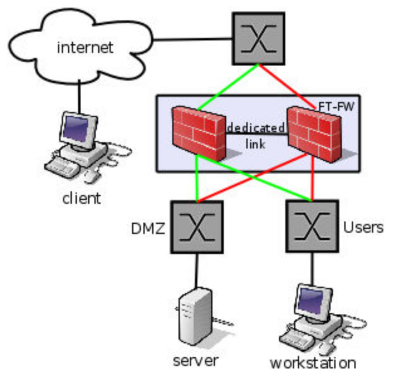

Fig. 1. Stateful firewall cluster of order 2 respectively

Definition 2. Cluster rule-set: Every replica firewall has the same rule-set.

Definition 3. Flow filtering: A stateful firewall fw_x filters a set of flows $F_x = \{F_1, F_2, ..., F_q\}$.

Definition 4. Multiprimary cluster: We assume that one or more firewall replicas deploy the filtering at the same time, the so-called *primary replicas*, while others act as *backup replicas*.

Definition 5. Failure detection: We assume a failure detection manager, eg. an implementation of VRRP [17] [18], that detects failures by means of heartbeat tokens. Basically, the replicas send a heartbeat token to each other every t seconds, if one of the replicas stops sending the heartbeat token, it is supposed to be in failure. This failure detection mechanism is complemented with several multilayer checkings such as link status detection and checksumming. This manager is also responsible of selecting which replica runs as primary and which one acts as backup. Also, we assume that the manager runs on every replica firewall belonging the cluster.

Definition 6: Flow durability (FD): The FD is the probability that a flow has to survive failures. If FD is 1 the replica firewall can recover all the existing flows. In this work, we introduce a trade-off between the FD and the performance requirements of cluster-based stateful firewalls.

Definition 7. Flow state: Every flow $F_i \in F$ is in a state s_k^i in an instant of time t_k.

Definition 8. State determinism: The flow states are a finite set of deterministic states $S = \{s_1, s_2, ..., s_n\}$.

Definition 9. Maximum state lifetime: Every state s_k^i has a maximum lifetime T_k. If the state s_k^i reaches the maximum lifetime T_k, we consider that the flow F_i is not behaving as expected, eg. one of the peers has shutdown due to a power failure without closing the flow appropriately.

Definition 10. State variables: Every state s_k^i is composed of a finite sets of variables $s_k^i = \{v_1, v_2, ..., v_j\}$. The change of the value of a certain variable v_a may trigger a state change $s_k^i \rightarrow s_{k+1}^i$.

Definition 11. State history: The backup replica does not have to store the complete state history $s_1^i \rightarrow s_2^i \rightarrow ... \rightarrow s_k^i$ to reach the consistent state s_k^i. Thus, the backup only has to know the last state s_k^i to recover the flow F_i.

Definition 12. State classification: The set of states S can be classified in two subsets: transitional and stable states. These subsets are useful to notice if the effort required to replicate one state change is worthwhile or not:

- *Transitional states* (TS) are those that are likely to be superseded by another state change in short time. Thus, TS have a very short lifetime.
- *Stable States* (SS) are long standing states (the opposite of TS).

We have formalized this state classification as the function of the probability (P) of the event of a state change (X). Let t be the current state age. Let T_k be the maximum lifetime of a certain state. Given the flow F_j in the current state s_k^j, we define the probability P_x that a TS can be superseded by another state change can be expressed as:

$$P_x(t, s_k^j) = \begin{cases} 1 - \delta(t, s_k^j) & if \quad (0 \leq t < T_k) \\ 0 & if \quad (t \geq T_k) \end{cases}$$

And the probability P_y that a SS can be superseded by a state change can be expressed as:

$$P_y(t, s_k^j) = 1 - P_x(t, s_k^j)$$

This formalization is a representation of the probability that a state can be replaced by another state as time goes by. Both definitions depend on the $\delta(t, s_k^j)$ function that determines how the probability of a state change s_k^j increases, e.g. linearly, exponential, etc. The states can behave as SS or TS depending on their nature, eg. in the case of TCP flows (Fig. 2), the initial handshake and closure packets (*SYN, SYN-ACK* and *FIN, FIN-ACK, ACK* respectively) trigger TS and *ACK* after *SYN-ACK* triggers TCP Established state which usually behaves as SS.

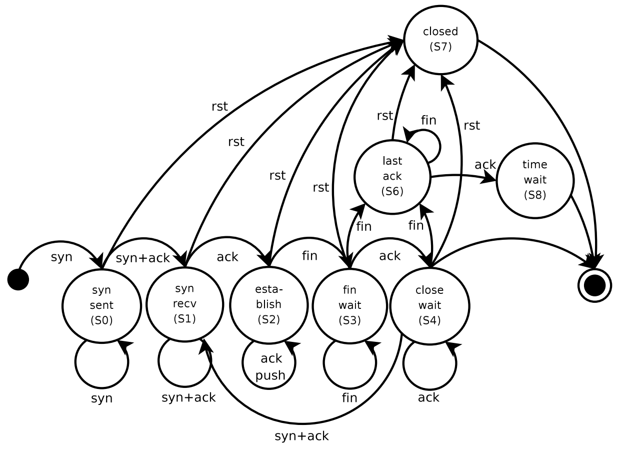

Fig. 2. A valid TCP flow state-machine extracted from Linux's Netfilter

Network latency (measured in round-trip time, RTT) is another important factor because if latency is high, all the states tend to behave as SS. In practise, we can define a simple $\delta(t, s_k^j)$ that depends on the acceptable network latency l:

$$\delta_x(t, s_k^j) = \begin{cases} 1 & if \quad t > (2 * l) \\ 0 & if \quad t \leq (2 * l) \end{cases}$$

The acceptable network latency l depends on the communication technology, eg. on a wired line the acceptable latency is 100ms and in satellite links 250 ms.

For the aim of this work, we focus on ensuring the durability of SS as they have a more significant impact on the probability that a flow can survive failures. This means that our main concern is to ensure that long standing flows can survive failures because the interruption of these flows lead to several problems such as:

1) Extra monetary cost for an organization, eg. if the VoIP communications are disrupted, the users would have to be re-called with the resulting extra cost.
2) Multimedia streaming applications breakage, eg. Internet video and radio broadcasting disruptions.
3) Remote control utility breakage, eg. SSH connections closure.
4) The interruption of a big data transfer between two peers, eg. peer to peer bulk downloads.

Nevertheless, the high durability of TS is also desired; however, they are less important than SS since their influence on the FD is smaller.

IV. Hybrid fault-tolerant architecture

Our proposed solution hybridizes both proactive and reactive fault-tolerant solutions. The architecture consists of two parts:

1) The proactive fault-avoidance implements a fault detector that triggers the migration of the flow-states from the unhealthy firewall replica to a healthy one. This migration happens if an error occurs, otherwise the fault-tolerant solutions does not consume any resource, thus, increasing the overall performance.
2) The reactive fault-handling propagates state-changes from the primary firewall replica to the backup firewall replica. However, the replication adopts a lazy approach as we do not propagate every state-change. Instead, the approach only replicates a subset of them that are considered to be SS.

Our solution supports the following scenarios:

1) A detectable error, ie. an error that the proactive part can recognize, triggers the migration of the states to a healthy firewall replica. Thus, the 100% of the flows are fully recovered.
2) A detectable error triggers the migration of the states to a healthy firewall replica. However, in the middle of the migration, the unhealthy node crashes. In this case, the proactive part only guarantees that a part of the flows will be recovered. Nevertheless, the lazy reactive approach gives us the chance to recover those flows that were not successfully recovered by the proactive part.
3) An undetectable error, ie. an error that the proactive part cannot recognize, results in a failure that crashes the node. Thus, the proactive part did not have any chance to initiate the migration of the flow-states. However, the lazy reactive approach gives us the chance to recover the flows that were not successfully migrated.

Thus, with regards to the reactive part, we have to define an approach that:

1) consumes few computational and bandwidth resources.
2) ensures that a high rate of flows can be recovered in case that the proactive part does not successfully migrate the states.

In the following sections we detail the proactive (Sect. V) and the reactive (Sect. VI) parts that compose our proposed architecture keeping in mind the previous key ideas to build a lazy reactive approach.

V. Proactive: Migration on errors

We assume that our cluster-based stateful firewall implements a proactive fault-tolerant framework similar to the one described in [13]. This includes a fault-tolerant detector and predictor (FPD) which uses some existing fault-detection framework such as [11] to detect, correct and notify PCI bus and RAM memory errors. Moreover, the FPD can also implement other fault detection techniques based on hardware sensors information [12] and log-files.

The FPD notifies to the policy daemon (PD) that an error of a certain seriousness will happen. The PD evaluates the error and, if it considers that it can compromise the firewalling, it tells the state migrator (SM) to extract the state-information of the existing flows from the stateful firewall and to propagate them to a sane firewall replica.

To ensure that the proactive approach works, we have to make the following assumptions:

1) Let T_{prop} be the time required to propagate the current flow-states plus the time required by PD to decide the appropriate action in answer to the error. Let T_{fail} be the time between an error occurrence and the failure time. Then T_{prop} must fulfill $T_{prop} < T_{fail}$ to ensure high flow durability. If this assumption is not fulfilled, we cannot ensure that the state migration will be succesfully accomplished.
2) If the evaluation of a certain error is not appropriate, the state migration may possibly not be successful. In other words, if the detection and evaluation of the error takes too long, then the necessary time T_{prop} cannot be assured. For that reason, the PD must deploy simple but conservative policies and implement fast runtime decision algorithms at the same time.

In spite of the detailed limitations, this approach does not incur any penalty in the system performance as the migration only happens when faults arise, and we assume that faults rarely occur. Still, we have to consider that it is not possible to predict and detect all possible errors before the system crash.

VI. Reactive: Bayes-based lazy replication

State replication is generally a resource-consuming task. This is particularly true if state-changes occur quite often and the number of replication messages become high. This reduces the overall scalability of the replication solution.

For that reason, we propose an improvement which consists of a simple routine that determines in runtime if the effort to replicate the state-changes s_k^i of a given flow F_i is worthwhile or not. For that purpose, we use Naive Bayes techniques that automatically learn the behaviour of the flow communications to decide if the resources invested to replicate s_k^i substantially improve the *flow durability* (FD), which is the probability that a flow can be successfully recovered.

The naive Bayesian classifiers provide a simple way to classify information in runtime that gives usually good results in practise, eg. in spam filters. Let C be the class, let X_i be the variable that represents one of the features of the system. Given a specific instance $x = \{x_1, x_2, ..., x_n\}$ of feature variables. A Bayesian classifier calculates the probability $P(C = c_k|X = x)$ for every possible class c_k as:

$$p(C = c_k|X = x) = \frac{P(X = x|C = c_k) * P(C = c_k)}{P(X = x)}$$

In this equation, $P(X = x|C = c_k)$ is the most difficult part to calculate. Naive Bayes [19] solves the problem in the most restrictive form by assuming that every variable x_i is conditionally independent of every other feature x_j given the class C. Therefore:

$$P(X = x|C = c_k) = \prod_i P(X = x_i|C = c_k)$$

A. State classification

We have classified state-changes in two types: transitional and stable. This classification can be used to decide if the effort required to replicate one state-change is worthwhile or not. Roughly speaking, *transitional states* (TS) are those that are likely to be superseded by another state-change in short time. Thus, TS have a very short lifetime. On the other hand, *Stable States* (SS) are long standing states so that they are the opposite of TS. We have formalized this state classification in the system model (Sect. III). This classification is simple but it provides good results in practise for online state classification.

B. Bayesian state replication

We assume that there is a knowledge base indexed by the tuple $Q_r = [Addr_{src}, Addr_{dst}, Port_{dst}]$ which is initially empty. Note that two different established flows F_i and F_j between peer p_A and p_B are identified by the same tuple Q_r if F_i and F_j refer to the same $Port_{dst}$. At startup, we assume that the base of knowledge is empty. Thus, when a flow F_i is filtered for first time a new tuple Q_r is inserted.

Every tuple Q_r points to one array A_r which stores the probability that the states can be transitional $A_r = \{P(TS|s_1) = v_1/t_1, ..., P(TS|s_n) = v_n/t_n\ \}$ where v_k identifies the number of TS out of the total t_k states s_n observed. Thus, the probability $P(TS|s_k) = v_k/t_k$ is calculated by observing the lifetime of state change s_k of every flow F_j in runtime. If the lifetime of the state change s_k goes over the classification barrier, we assume that it behaves as TS, therefore $P(TS|s_k) = v_k + 1/t_k + 1$, otherwise $P(TS|s_k) = v_k/t_k + 1$.

The $Port_{dst}$ parameter is useful since the same peer p_B can provide different services to the same peer p_A, eg. p_B can provide HTTP at port 80 and SSH at port 22, and the service provider may have applied different Quality of Service (QoS) policies depending on the type of service like prioritizing interactive traffic can be over bulk traffic. Thus, the set of TS and SS for SSH and HTTP flows may differ due to QoS.

We define the domain of the class C into transitional and stable states, and the set of features is the domain of possible flow states $S = \{s_1, s_2, ...s_n\}$. Thus, the probability that a certain state s_k is a transitional state is:

$$P(C = TS|X = s_k) = \frac{P(X = s_k|C = TS) * P(C = TS)}{P(X = s_k)}$$

Similarly, the probability that a state s_k is a stable state is:

$$P(C = SS|X = s_k) = \frac{P(X = s_k|C = SS) * P(C = SS)}{P(X = s_k)}$$

Intuitively, we deduce that:

$$P(C = SS|X = s_k) + P(C = TS|X = s_k) = 1$$

In order to resolve these equations, we have to define a training period where we collect how every state-change belonging S behaves by using the classification barrier. Thus, we define a simple solution to reduce the number of replication messages: if the state-change s_k behaves as TS, we delay or eliminate the replication. On the other hand, if the state-change represents a SS, it is propagated to the neighbour mesh AP.

As the latency of the network may change due to several factors, the state-change prediction may start failing. In order to solve this problem, we can define a maximum number of failing predictions after which we reset the outdated tuple Q_r.

VII. Evaluation

In order to evaluate our approach, we have deployed a simple Primary-Backup cluster-based stateful firewall of order two based on our free-software daemon for Linux [20] in a segment of the university network. The setup is composed of Intel Xeon 1.6GHz with 1GEthernet cards. We have implemented an extension in C to enable the Naive Bayes-based replication that has around 400 LOC [21]. We did not enable any compiler optimization in our experiments. We have emulated the users behaviour with a small bot that: connects to HTTP websites in France, Spain, and Japan; checks for new messages via POP3 and IMAP; file downloads via FTP; and starts SSH connections with several servers in Spain and France. The size of the experiment is small as we have only measured the behaviour of our implementation until we have observed the end of 200 flows. We plan to perform bigger experiments in the near future.

We aimed to evaluate the following aspects that are important to validate our proposed solution:

1) The time required by the proactive approach to migrate the states from one hypothetic unhealthy firewall replica to a healthy one.
2) The number of state-changes that can be saved by the Bayes-based reactive lazy replication, including the good and wrong prediction rates.
3) The amount of computational resources invested in the Bayes-based reactive lazy replication.

We do not cover further evaluation of the proactive part as we consider that the formalization of a proactive framework is out of the scope of this work.

We have evaluated the extra CPU and memory consumption of the Bayes-based lazy replication. Our experiments have shown that the CPU consumption is negligible to our previous works [8][9]. With regards to the memory consumption, our implementation consumes the 80 bytes per tuple T (see Sect. VI for details) to store the states behaviour.

A. Bayes-based replication: prediction with homogeneous network latency

For our experiments, and according to Sect. III, we have adopted a boundary of 200ms to train Naive Bayes, ie. if a state lives more than 200ms is a SS, otherwise it is a TS. The measured homogeneous network latency is around 30ms.

Moreover, as Naive Bayes approach returns the probability that a state s_k is SS (not a boolean answer), we have to select a barrier that is used to determine the prediction, eg. if the barrier is set to 0.05, Naive Bayes must return ≤ 0.05 to classify s_k as TS. We assume that a wrong prediction occurs when a SS is classified as TS since then s_k is not replicated.

Intuitively, we do the following observations: if we keep the barrier low, Naive Bayes easily predicts that most states are stable. Thus, we have less chances to save states in the replication but the chances to hit wrong predictions decreases.

On the other hand, if we keep the barrier high, Naive Bayes easily predicts that most states are transitional. Thus, we have more chances to save states in the replication but also the chances to hit wrong preditions increases.

Therefore, we have to look for a low barrier that gives a good tradeoff between a high rate of saved-states and a low rate of wrong predictions.

The following figures shows the experimental results when the barrier was set to 0.125 (Fig. 3), 0.25 (Fig. 4) and 0.50 (Fig. 5). The results initially may fluctuate since Naive Bayes requires some time to learn the state-change behaviour until it stabilizes.

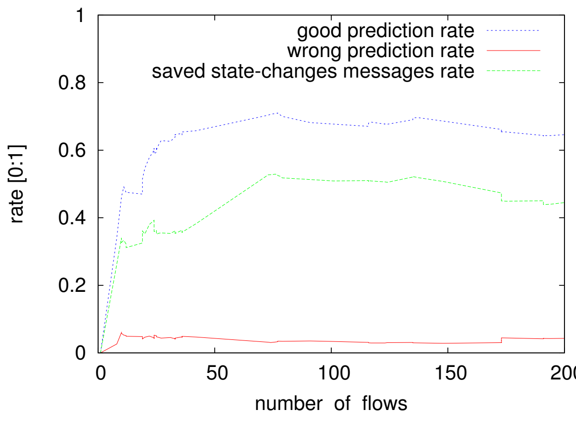

Fig. 3. Bayesian replication using barrier 0.125 - homogeneous latency

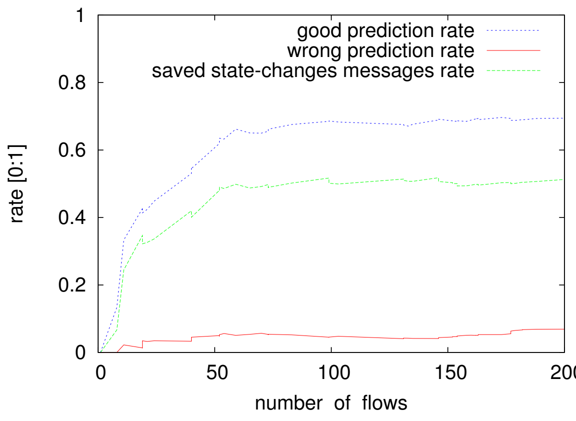

Fig. 4. Bayesian replication using barrier 0.25 - homogeneous latency

As we can observe, using 0.50 as barrier increases the number of saved states in the replication but dramatically increases the number of wrong predictions close to 18%. Thus, in case that the proactive part fails to migrate the service, the reactive part will not be able to recover 18% of flows.

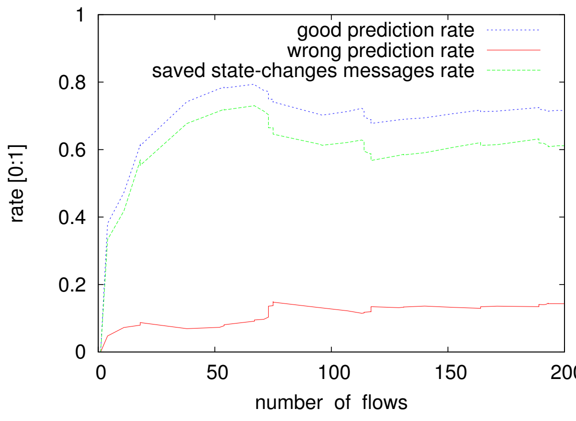

Fig. 5. Bayesian replication using barrier 0.50 - homogeneous latency

In Fig. 6, we have represented the evolution of the good predition rate, the wrong prediction rate and the saved state-changes rate. The results show that adopting a low barrier ensures a save of 50% in the state replication. The increase of the barrier consequently increases the wrong predictions. The use of a low barrier forces Bayes to have high certainty that the state is trancisional.

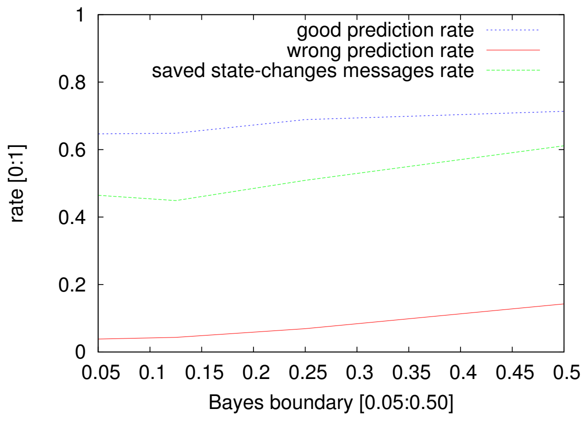

Fig. 6. Bayesian replication: barrier comparison results

B. Bayes-based replication: prediction with heterogeneous network latency

We have repeated the measurements by introducing heterogeneous latency in the network. Specifically, we have used a normal distribution of 20-500ms with Linux netem[1].

With regards to the previous experiments with an homogeneous network latency, we observe that the rate of wrong predictions increases quicker with heterogeneous network latency. In Fig. 8, the rate of wrong predictions using a barrier of 0.25 to classify states is close to 19% which is similar to the results obtained using a barrier of 0.50 with homogeneous

[1]http://www.linuxfoundation.org/en/Net:Netem

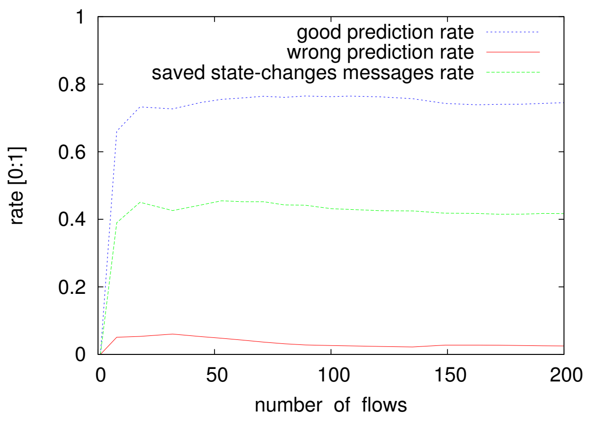

Fig. 7. Bayesian replication using barrier 0.125 - heterogeneous latency

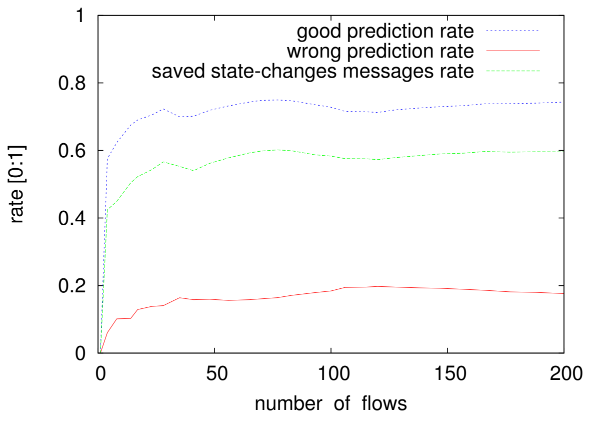

Fig. 8. Bayesian replication using barrier 0.25 - heterogeneous latency

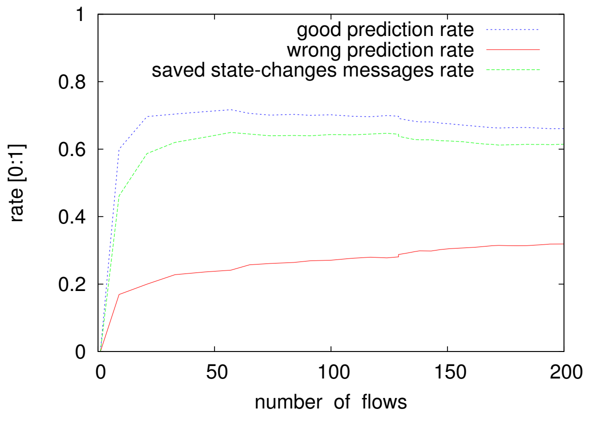

Fig. 9. Bayesian replication using barrier 0.35 - heterogeneous latency

latency. However, the number of saved-states is around 40% using a barrier of 0.125 (Fig. 7) which is still a good rate.

Thus, we can conclude that keeping the barrier low ensures a save around 40-50% while the rate of wrong predictions does not goes over 2%.

In Fig. 10, we represent the relationship between the good prediction rates, the wrong prediction rates and the saved-states in a heterogeneous network.

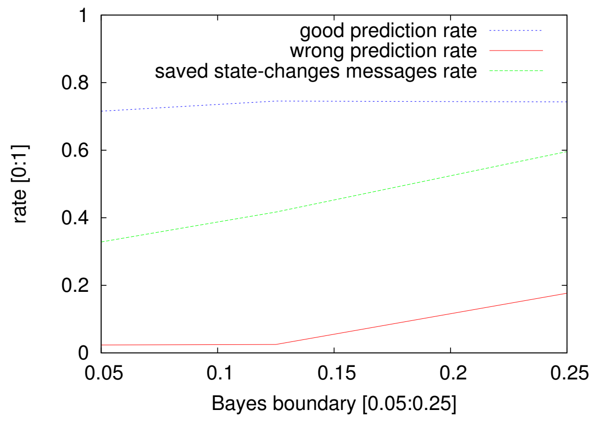

Fig. 10. Bayesian replication: barrier comparison results

C. Proactive migration

We have measured and normalized the time required to migrate the flow states between two replica firewalls. In our experiments, we have emulated different message omission rates. Of course, we assume the use of the reliable replication protocol detailed in [8]. The results show that the time to transfer the states between two replicas with no packet loss are around 4 seconds for 25000 state-flows. This time increases if there is message omission situations, as the backup firewall replica requests retransmissions to the primary firewall replica.

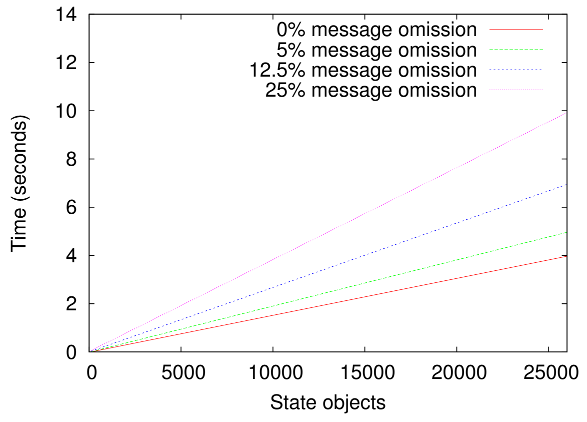

Fig. 11. Proactive migration time

VIII. Conclusion and future works

In this work, we extend our previous works by proposing an hybrid fault-tolerance approach for clusted-based stateful firewalls that mixes proactive fault-avoidance with lazy reactive fault-handling. This approach reduces the computational resource consumption with regards to our existing reactive solution, and it is less risky than a pure proactive approach.

Adopting a low barrier in Naive Bayes to classify states, our solution ensures that the reduction in the state replication is 40-50%, which is a considerable reduction, and the rate of wrong predictions is around 2%. The solution benefits from the proactive approach to reduce the resources involves in the state replication.

As future work, we plan to evaluate more sophisticated machine-learning approaches to classify states keeping in mind the performance runtime requirements of the covered scenario. We are also studying other Internet services with similar semantics that can benefit from this hybrid architecture.

Acknowledgment

This work has been partially supported by the Spanish Ministerio de Educación y Ciencia through a coordinated research project(grant DPI2006-15476-C02-00) and Feder (ERDF).

References

[1] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, “Understanding replication in databases and distributed systems,” *ICDCS*, 2000.

[2] R. Jimenez-Peris, M. Patino-Martinez, B. Kemme, and G. Alonso, “How to select a replication protocol according to scalability, availability, and communication overhead,” *SRDS*, p. 24, 2001.

[3] R. Zhang, T. Adelzaher, and J. Stankovic, “Efficient TCP Connection Failover in Web Server Cluster,” in *IEEE INFOCOM 2004*, Hong Kong, China, mar 2004.

[4] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode, “Migratory TCP: Connection Migration for Service Continuity in the Internet,” in *22nd International Conference on Distributed Computing Systems*, 2002.

[5] M. Marwah, S. Mishra, and C. Fetzer, “TCP server fault tolerance using connection migration to a backup server,” *In Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN), pages 373–382, San Francisco, California, USA*, Jun 2003.

[6] N. Ayari, D. Barbaron, L. Lefevre, and P. Primet, “T2CP-AR: A system for Transparent TCP Active Replication,” in *AINA ’07: Proceedings of the 21st International Conference on Advanced Networking and Applications*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 648–655.

[7] A. Gorti, “A faul tolerant VoIP implementation based on Open Standards,” in *IEEE Proceedings EDCC-6, pag. 35-38, Coimbra, Portugal*, Oct. 2006.

[8] P. N. Ayuso, R. M. Gasca, and L. Lefevre, “FT-FW: Efficient Connection Failover in Cluster-based Stateful Firewalls,” in *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 573–580.

[9] P. Neira Ayuso, L. Lefevre, and R. M. Gasca, “Multiprimary support for the availability of cluster-based stateful firewalls using FT-FW,” in *Proceedings ESORICS’08: European Symposium on Research in Computer Security, Malaga, Spain*, Oct. 2008.

[10] P. Narasimhan, T. A. Dumitras, A. M. Paulos, S. M. Pertet, C. F. Reverte, J. G. Slember, and D. Srivastava, “Mead: support for real-time fault-tolerant corba: Research articles,” *Concurr. Comput. : Pract. Exper.*, vol. 17, no. 12, pp. 1527–1545, 2005.

[11] “Linux edac (error detection and correction) project,” http://bluesmoke.sourceforge.net.

[12] N. A. D. Turnbull, “Failure prediction in hardware systems,” Departament of Computer Science. University of California, CA, USA, Tech. Rep., 2003.

[13] G. Vallee and K. Charoenpornwattana, “A framework for proactive fault tolerance,” in *Proceedings of 3rd international conference on Availability, Reliability and Security (ARES)*, Mar. 2008.

[14] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, “Proactive fault tolerance for hpc with xen virtualization,” in *ICS ’07: Proceedings of the 21st annual international conference on Supercomputing*. New York, NY, USA: ACM, 2007, pp. 23–32.

[15] G. Vallee, T. Naughton, C. Engelmann, H. Ong, and S. L. Scott, “System-level virtualization for high performance computing,” *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008*, vol. 0, pp. 636–643, 2008.

[16] M. Gouda and A. Liu, “A model of stateful firewalls and its properties,” *Proceedings of International Conference on Dependable Systems and Networks, 2005 (DSN)*, pp. 128–137, 28 June-1 July 2005.

[17] R. Hinden, “Rfc 3768: Virtual router redundancy protocol (vrrp),” apr 2004.

[18] A. Cassen, “Keepalived: Health checking for lvs & high availability,” http://www.linuxvirtualserver.org.

[19] I. Good, “The estimation of probabilities: An essay on modern bayesian methods,” *MIT press*, 1965.

[20] P. Neira, “conntrackd: The netfilter’s connection tracking userspace daemon,” http://people.netfilter.org/pablo/.

[21] ——, “Extension to implement naive bayes state replication,” 2008. [Online]. Available: http://dune.lsi.us.es/bayes.tgz