FT-FW: a cluster-based fault-tolerant architecture for stateful firewalls

Pablo Neira Ayuso[a,*], Rafael M. Gasca[a], Laurent Lefevre[b]

[a] *QUIVIR research group, Department of Computer Languages and Systems, ETS Ingenieria Informatica, Avda. Reina Mercedes, s/n, 41012, Sevilla, Spain*

[b] *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*

Abstract

Nowadays, stateful firewalls are part of the critical infrastructure of the Internet. Basically, they help to protect network services and users against attackers by means of access control and protocol conformance checkings. However, stateful firewalls are problematic from the fault-tolerance perspective since they introduce a single point of failure in the network schema. In this work, we summarize and enhance our previous research efforts that aim to provide a full fault-tolerant solution for stateful firewalls. These efforts have focused on the design and the implementation of the cluster-based Fault-Tolerant stateful Firewall (FT-FW) architecture. We provide details on our proposed solution and we extensively evaluate important network performance and availability aspects that we did not cover so far. The evaluation experiments are based on our Free/OpenSource implementation that has become the most popular solution for Linux-based stateful firewalls [1].

Keywords: stateful firewall, fault-tolerance, computer security, computer networks, access control

1. Introduction

Firewalls are a key part of the Internet infrastructure to protect users and network services against attackers. Basically, a firewall separates several network segments and it enforces a filtering policy. This filtering policy determines what packets are allowed to enter and to leave a given network segment. The filtering policy is expressed in an Access Control List (ACL) that uses a vendor-specific low-level language. The ACL contains a list of rules where each rule contains one or several packet selectors, eg. the source and destination address, port source and destination, protocol type, and other fields available in the packet headers; and one action, eg. accept, deny, log among many others.

Stateful firewalls (Gouda and Liu, 2005) extend packet filtering capabitities by performing conformance checkings upon the network traffic. Basically, they enforce that the communications between two peers evolve according to the protocol specification. In practise, stateful firewalls implement a finite state automaton for each supported protocol that determines what state-transitions are valid from the current state. Then, for each network packet, stateful firewalls check if it triggers a valid state-transition.

This stateful capability allows the firewall administrator to modify the ACL to perform some action on the packets that result in invalid state-transitions, like logging and dropping them. Stateful firewalls also support application layer inspection to assist the filtering of popular multi-session application protocols like FTP, H.323 and SIP.

From the fault-tolerance perspective, firewalls introduce a single point of failure in the network schema. Thus, a failure in the firewall results in the temporary isolation of the network segments during reparation. According to empirical caracterizations of hardware failures in cloud computing (Vishwanath and Nagappan, 2010), the chances to see any hardware failure is low during the initial 3-5 years of lifetime. In the aforementioned study, the annual failure

[1] This includes commercial firewall vendors that base their products on Linux and OpenSource software like Vyatta Inc. (http://www.vyatta.com), Astaro AG (http://www.astaro.com) and 6WIND S.A.R.L. (http://www.6wind.com). According to the information available in their websites, these vendors sell their products to SMEs, Fortune 50 companies and the public administration all over the world.

* Corresponding author.

Email addresses: pneira@us.es (Pablo Neira Ayuso), gasca@us.es (Rafael M. Gasca), laurent.lefevre@inria.fr (Laurent Lefevre)

rate for servers is 8%, where 70% of all failures are due to hard disk failures and 5% due to memory. From these results, we can deduce that firewalls that are based on commercial off-the-shelf hardware (COTS) are unlikely to fail. Some firewall vendors have decided to replace hard disks by compact flash in their appliances, thus, failures become even more unlikely since hard disks seem to be the main source of problems. Nonetheless, network designers have to evaluate the impact of unexpected downtime due to firewall failures depending on the Service Level Agreements (SLA). In the particular case of tight SLAs, that may specify extremely high availability requirements[2], firewall failures may easily lead to violating these agreements.

Traditionally, fault-tolerance has been addressed by means of physical redundancy and health check techniques. Basically, the idea consists of deploying a cluster that is composed of two or more stateful firewalls: at least one that filters flows (primary), and others (backups) that are ready to replace the primary if it experiences a failure. However, physical redundancy is insufficient for stateful firewalls, more specifically if you want to achieve zero downtime. Since the stateful filtering depends on the flow-state, we have to ensure that the existing flow-states survive accross failures. In a nutshell, the solution for this problem consists of propagating the existing flow-states from the primary to the backup firewalls. However, we have to take into consideration the stateful firewall properties to provide a feasible fault-tolerant solution.

In this work, we summarize, enhance and update our previous works on fault-tolerant stateful firewalls by providing: more in-depth details on our architecture, an elaborated description of our replication protocol which has been improved since our first proposal, discussion on several interesting aspects of the solution and extensive experimental evaluation of our implementation which remained preliminary.

This article is organized as follows: In section 2 we cover the state of art. Then, in section 3 we provide the system model for stateful firewalls and the problem description. Section 4 presents the FT-FW architecture and section 5 details our proposed replication protocol. We discuss several design and implementation decisions of FT-FW comparing them to those of other existing solutions in section 6. The experiments and evaluation are exposed in section 7. We extend our experiments by providing experimental results in one real-world deployment in section 8. Then, we conclude this paper in section 9 with future work and conclusions.

2. Related work

Firewalls introduce several problems that have been addressed by an interesting amount of research literature during the last decades. We have grouped these problems into three categories: performance, complexity and fault-tolerance. Due to the nature of this work, we specifically focus on fault-tolerance related works.

2.1. Network performance

Firewalls may degrade latency and bandwidth throughput. The problem is that firewalls receives packets that have to be checked against the ACL that is expressed in a list of rules. Thus, the time required to match packets against a rule grows in different orders depending on the algorithm and data structure that is used (Pozo et al., 2010). This is specifically an important problem for large ACLs. Moreover, performance packet-matching algorithms usually require complex data structures that consume a lot of memory or specialized hardware. Therefore, a bad packet-matching approach and a large ACL may severely harm network performance.

2.2. Complexity

Firewalls are complex solutions that have to be configured in order to administer network resources appropriately (Wool, 2004). Basically, the problem is that firewall ACLs are expressed in vendor-specific low-level languages. Thus, the writing of an ACL remains a difficult task that tend to be error-prone (Wool, 2004). Moreover, if rules are expressed using wildcards that allow to filter entire subnets instead of single addresses, they may not be disjoint. Thus, rule ordering becomes important since overlappings may lead to consistency problems. On the other hand, building a consistent hierarchy of distributed firewalls is also a challenging task (Pozo et al., 2010) (Al-Shaer et al., 2005) (Bartal et al., 2004), even more if it must support frequent ACL updates (Al-Shaer and Hamed, 2006).

[2] e.g. 99.999% of availability, which means 5.26 minutes of downtime per year

2.3. Fault tolerance

Nowadays, you can find open and closed-source fault-tolerant stateful firewall products from different vendors. In general, there are few internal details on how closed-source products solve the fault-tolerance problem. The existing information mainly describes how to install and to configure the fault-tolerant setups for these products.

In the closed-source domain, CheckPoint Firewall-1 offers state-replication since the 2000s. They use a Multicast UDP protocol over port 8116 that operates in one of these two modes:

- Full synchronization, that is used after the reboot of one firewall to obtain all the flow-states from the primary firewall.
- Incremental synchronization, that consists of one batch with all flow-state changes. This batch is sent every 100ms.

The possible cluster configurations are primary-backup and multiprimary with load-sharing support. You have to deploy a couple of firewalls running the same operating system and software versions. Thus, software diversity is not allowed. State-replication is not recommended if the rate of session/s is high (Welch-Abernathy, 2004).

Cisco PIX/ASA (Cisco Systems Inc., 2011) also supports fault-tolerant setups for their stateful firewalls. It allows regular (stateless) failover and stateful failover. According to the documentation, firewalls continually pass flow-state updates. You can deploy multiprimary setups and you have to use twin devices using the same software versions as well.

With regards to the Free software/OpenSource Software (FOSS) community, the OpenBSD project provides the so-called *PFSync* (McBride, 2004). This solution allows to replicate flow-states between a cluster of stateful firewalls. The solution remains monolithic since it is embedded into the firewall source code that is executed in kernel-mode. Moreover, the replication protocol to propagate flow-states between the stateful firewalls is built upon unreliable Multicast IP. As for many FOSS projects, the most up-to-date reference remains the source code. There are some online articles that describe the recent re-design of the PFSync protocol (Gwynne, 2009). In these documents, the author explain how backward compatibility has been discontinued in order to improve the solution. For that reason, they recommend to use the same software version in both firewalls. There also exist other FOSS projects such as Linux-HA (Robertson, 1999), CARP (McBride, 2004) and Keepalived (Cassen, 2002) that provide redundancy protocols to health-check nodes in a cluster and to replace them under failures. These provide the basic building blocks to deploy stateless fault-tolerant firewalls.

In the academia field, researchers have proposed numerous fault-tolerant approaches for Internet network services such as web servers (Zhang et al., 2004), TCP-based back-end servers in general (Sultan et al., 2002) (Marwah et al., 2003) (Ayari et al., 2007), VoIP PBX (Gorti, 2006) and CORBA (Narasimhan et al., 2005), among many others. Basically, they solve the fault-tolerance problem by means of physical redundancy of general-purpose COTS equipments; and software-based approaches that are more resilient to changes in the services than hardware-specialized solutions (Powell, 1994). Off-the-shelf hardware is usually selected since they are relatively unexpensive to purchase and to maintain, and they provide an environment in which development time is reduced. These works are useful to address the stateful firewalls fault-tolerance problem. However, they cannot exploit interesting stateful firewall properties that we do in our work to provide a better solution. Moreover, most of the experiments in these works cover 10/100 Mbit networks since it was the predominant technology at that time (end of 90s). More recently, the research community have also proposed fault-tolerant solutions that rely on extra hardware (Sultan et al., 2005) and, even more recently, based-on virtualization techniques (Vallee and Charoenpornwattana, 2008) (Vallee et al., 2008) (Nagarajan et al., 2007). An article from one of the authors of this work (Ayari et al., 2008) summarizes the existing state of art in the area of fault-tolerant network equipments and services.

In our previous works in the FT-FW saga, we proposed an event-driven architecture and the first version of our replication protocol to build cluster-based single-primary fault-tolerant stateful firewalls (Neira-Ayuso et al., 2008a). We extended it to add preliminary support for multi-primary setups (Neira-Ayuso et al., 2008c). In these works, we provided superficial experimental evaluation that we enhance in this article. The main features of the solution that we provide in the aforementioned works are:

1. Simplicity. We reuse and extend existing software-based, high availability solutions. The client-side does not require any modification. Thus, this is a client transparent solution. The firewall requires minimal and non-intrusive modifications.

2. Performance. The solution ensures negligible delay in client responses and quick recovery from failures. Clients notice low performance drop. Our experimentation confirms that our proposed solution is suitable for 1 GEthernet network setups.
3. Low cost. The solution is suitable for COTS equipments and it requires no hardware extensions.
4. Multi-primary workload sharing setups. The proposed architecture supports advanced setups where several firewalls share workload to improve scalability.

We have also surveyed existing stateful firewall cluster configurations, we have described their challenges and we have provided evaluation results for the detailed configurations (Neira-Ayuso et al., 2009).

3. System Model and Problem Description

3.1. System Model

A formalized model for stateful firewalls and its properties has already been already proposed (Gouda and Liu, 2005). For that reason, it is not in the scope of this work to provide a full model. Instead, we provide the definitions, notations and properties that we consider useful for our needs.

We assume that a stateful firewall FW_x filters a set of flows $F = \{F_1, F_2, ..., F_n\}$ in an instant of time t. Every flow F_i in F is in a flow-state S_k. The flow-states are a finite set of deterministic states $s = \{S_1, S_2, ..., S_n\}$ and the flow-state transitions $S_n \rightarrow S_m$ are expressed in a finite state automaton for each supported protocol that determines what protocol transitions are valid (see Fig. 1 for reference). The maximum number of flows that can be filtered depends on the amount of memory that is dedicated to store the flow-states.

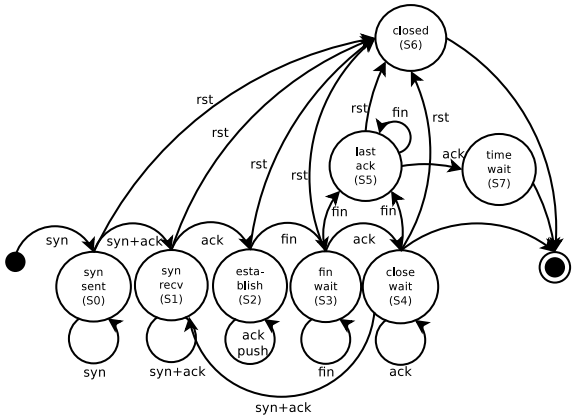

Figure 1: An example TCP state automaton extracted from Linux's stateful firewall.

Every state S_k is composed of a finite sets of variables $S_k = \{v_1, v_2, ..., v_j\}$. The change of the value of a certain variable v_a may trigger a state transition $S_n \rightarrow S_m$. Moreover, every state S_k has a maximum lifetime T_k. Thus, if the state S_k reaches the maximum lifetime T_k, we consider that the flow F_j is not behaving as expected, eg. one of the peers has vanished due to a power loss without closing the flow appropriately.

Flows and their flow-states are identified by means of the flow-tuple: $\{Address_{SRC}, Address_{DST}, Port_{SRC}, Port_{DST}, Protocol\}$. This flow-tuple is used as key to look up for a given flow-state. Basically, stateful firewalls build this flow-tuple from the network packet to find what flow-state matches this packet. Then, they can verify if that network packet triggers a valid state-transition.

We also assume that all network protocols are stateful, even those whose nature is purely stateless, such as UDP. For stateless protocols, we can define a simple state-automaton that consists of two states: a) we have only seen packets in one direction so the flow is not established, and b) we have seen packets in both directions so we can consider that the flow is established.

Our cluster-based setup is composed of a set of stateful firewalls $FW = \{FW_1, ..., FW_n\}$ where $n \geq 2$. The set of stateful firewalls FW are deployed in the local area network and they use the same ACL. We also assume that failures are independent, so the addition of new firewalls to the cluster improve availability.

To monitor failures of the set of stateful firewalls, we assume a *failure detection manager*. This manager runs on each stateful firewall FW_x to health-check itself and other stateful firewall nodes that belong to the cluster. The stateful firewall can be in two working modes: a) *Primary*, that means that it is active and performing the stateful filtering, and b) *Backup*, in which the firewall is waiting for the failure of one of the primaries to replace them. A stateful firewall FW_x can be in primary and backup working modes at the same time, meaning that it is deploying the stateful filtering but it is also acting as backup for another stateful firewall FW_y. If a stateful firewall experiences a failure, the failure detection manager selects what backup stateful firewall becomes primary to recover the stateful filtering.

Backup stateful firewalls do not have to store the complete flow-state history $S_1 \rightarrow S_2 \rightarrow ... \rightarrow S_k$ to reach the consistent flow-state S_k. Thus, the backup can recover the flow F_i as soon as it stores the current flow-state S_k for F_i. This is a specific property of the stateful firewall semantics that we exploit in our architecture. We call this property the *flow-state independency*.

3.2. Problem description

Without state-replication, the established flows may break. This may be particularly disturbing in several situations, for example:

1. Desktop experience: user applications may misbehave during the failover resulting in the the halting of multimedia applications, eg. popular youtube-like Internet video and radio streaming services; the disruption of remote control utilities, eg. SSH and remote desktop connections; and the interruption of a big data transfers between two peers, eg. bulk downloads that cannot be resumed and would need to be re-started.
2. Monetary cost: extra monetary cost for an organization, eg. if the VoIP communications are disrupted, your users have to re-call and you will have to pay the possible extra cost due to the call re-establishment. Another possibility is the violation of the SLA, that may result in decreasing the customer's confidence on you and, at worst case, losing that customer because he decided to migrate to a different service provider that provides better availability.
3. Firewall upgrade and maintainance tasks: network administrators can schedule downtime for one of the firewalls that belong to the cluster to perform upgrade and maintainance operations without the risk of disturbing existing communications. In terms of security, software updates are frequent to resolve exploitable software bugs. Moreover, some of these updates may require rebooting the firewall. Generally speaking, this may result in unavailability time that goes from seconds to a couple of minutes depending on how fast the firewall restarts.

More specifically, we describe the problem assuming a scenario that is composed of one client, one server and one cluster of stateful firewalls. For simplicity, the cluster is composed of two stateful firewalls, one that runs as primary and the other as backup. Both stateful firewalls implement the TCP state-automaton that we have represented in Fig. 1.

Initially, the client starts a bulk download over TCP with a server that is behind the cluster of stateful firewalls. The client performs the classic three-way TCP-handshake that goes accross the primary firewall to the server. Thus, the TCP-handshake triggers several valid TCP-state transitions to the *TCP Established* state in the primary stateful firewall. Then, the primary experiences a failure so the backup firewall becomes primary to recover the filtering. However, the first packet that the new primary firewall sees is a PUSH packet that is part of the established TCP connection. Thus, it triggers an invalid transition according to the TCP state-automaton and the flow is discarded.

One possible solution for this, also known as the *poor man* approach (Welte, 2004), consists of disabling stateful firewalling after the failover and to enable some sort of flow-state pickup facility during some time. This facility allows to infer the existing flow-states from the traffic that the firewall is accepting at that moment. After some specific amount of time[3], the firewall enters back to the stateful filtering mode. This simplistic approach is problematic since it allows one attacker to inject bogus flow-states during the failover. Basically, if the attacker can deploy a Denial-of-Service attack to make your primary firewall fail, it can temporarily disable the conformance checking that the stateful firewall performs. Moreover, the attacker has the chance to send forged packets during the failover to fool the state tracking. These evil packets can be used to inject fake flow-states to access protected servers behind the

[3]This should be a configurable parameter. If the selected amount of time for the picking up is small, many flows may not be recovered appropriately. On the other hand, if it is high, you reduce security since conformance checkings will be disabled for long time.

firewall. On the other hand, if the pickup time is too small, some existing traffic may break. Specifically, if no packets are seen for some established communication during the pickup, the firewall will not have any flow-state information about it. Thus, this communication is likely to break once the stateful filtering is enabled again. For these reasons, we discourage the use of this approach.

The key idea that we propose to avoid the breakage of existing communications is state-replication. The flow-states need to be propagated to the backup firewalls to allow an appropriate recovery of the existing flows. With state-replication, we ensure that all flow-states (or a sub-set of them) are propagated to other stateful firewall nodes that are part of the cluster. Basically, the primary stateful firewall propagates state-changes to the backup stateful firewalls. Thus, if the primary fails, the selected backup to become the new primary can successfully recover the filtering.

4. FT-FW architecture

The architecture that we propose for fault-tolerant stateful firewall have both hardware and software aspects that we detail in the following subsections.

4.1. Hardware requirements

Our architecture, from the hardware perspective, is composed of a set of stateful firewall that are part of the cluster. We use COTS hardware as firewalls, which seems to be the main choice in cloud computing (Vishwanath and Nagappan, 2010). These stateful firewalls are deployed in the local area network. The selection of the number of stateful firewalls belonging the cluster depends on:

a) Availability: assuming that failures are independent, increasing the redundancy degree, ie. adding more stateful firewall nodes, improves availability.

a) Performance: if the appropriate configuration is adopted, we can improve performance scalability by sharing the workload between the stateful firewalls that are part of the cluster.

a) Complexity: adding more stateful firewall nodes incurs in some extra complexity in the network design and maintainance.

We assume that the stateful firewall nodes are connected through one or more dedicated links (alternatively, you may use some secured channel, e.g. IPSec). These dedicated links are used to propagate flow-state changes and to health-check stateful firewall, as we further detail in this work.

More specifically, we support two different redundancy-based architectural designs for stateful firewalls that provide a good trade-offs between availability, performance and complexity, they are:

a) Primary-Backup, in which one firewall is actively filtering flows and the backups are idle. If the primary fails, one of the backup firewall becomes the new primary.

a) Multi-primary load-sharing, in which all the firewalls are actively filtering flows. The firewalls receive all the same network packets in a multicast-basis but only one performs the filtering based on a per-flow hash-based approach. If one firewall fails, the flows are recovered by one of the primaries.

We provide more extensive details on the architectural design of these two setups in one of our previous works (Neira-Ayuso et al., 2009).

Link redundancy is an important aspect to achieve fault-tolerance in networks since they are also single point of failures in the network schema. This problem can be addressed with link aggregation (IEEE 802.1AX-2008, formerly IEEE 802.3ad) and redundant ISP providers. Redundancy in links ensures that if one fails, we can failover to another. We do not cover link redundancy aspects since we consider that it is not in the scope of this work.

4.2. Software blocks

We propose an architecture that is composed of three software blocks. We assume that there is an instance of each software block in every stateful firewall node that belongs to the cluster, they are:

a) the Connection Tracking System (CTS) (Neira-Ayuso, 2006b), that performs the conformance checkings upon the protocols that the firewall filters. Basically, this is the core software block of the stateful firewall. Without it, no stateful filtering can be performed.
b) the State-Proxy (SP), that is a process that receives state-changes from the CTS and it propagates them to other SPs that are running in other firewall nodes that belong to the cluster.
c) the Failure Detection Manager (FDM), that is a process that monitors and health-checks the firewall nodes that belong to the cluster and it assists the SP on what to do in the presence of failures. The FDMs also select what backup stateful firewall node becomes the newly elected primary under failures.

Our CTS provides three methods that can be used by the SP to interact each other, that are:

a) a subscription method that allows the SP to track all state-changes or a given sub-set of them.
b) an export method that the SP can use to retrieve existing flow-states living in the CTS.
c) an import method that the SP can use to push flow-states into the CTS.

Since one SP may also receive state-changes from another, the SP has two choices on what to do with these state-changes, they are:

a) direct flow-state injection: they can straight-forward inject the flow-state into the CTS.
b) cache-based storage: they can store the foreign flow-states in a cache that would be imported into the CTS under failures.

The interaction between the SP and the CTS follows an event-approach that allows the SP to obtain the existing flow-states from the CTS reliably. Basically, the CTS provides methods that the SP can use to subscribe to state-changes notifications. The propagation of state-changes is performed in a soft real-time basis, ie. state-changes are delivered as soon as possible.

The FDMs monitor all the firewall nodes that are part of the cluster. Basically, they detect stopping failures by means of heartbeat tokens. FDMs send these heartbeat tokens to each other every t seconds. If one of the FDM stops sending the heartbeat token, it is assumed to be in failure. This failure detection mechanism is complemented with several multilayer checkings such as link status detection, checksumming and process monitoring. Thus, if a firewall node experiences a failure, the FDMs select which firewall node recovers the filtering and it informs the SP about the situation. Under failure, the SP invokes the import method to inject the flow-states into the CTS.

We have represented the interaction between the different software blocks that compose our architecture in Fig. 2.

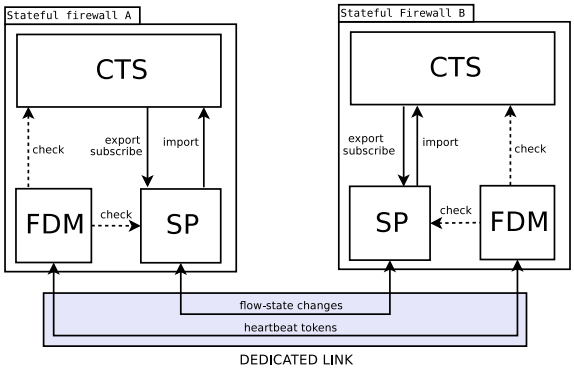

Figure 2: Software blocks and their interactions.

SPs use the FT-FW protocol to communicate each other. This is an asynchronous and reliable protocol that we detail in Sect. 5.

Our architecture allows the system administrator to add support for fault tolerance in a plug-and-play fashion. Basically, the system administrator only has to launch the SP in run-time on each existing stateful firewall that belong to the cluster to improve availability.

5. FT-FW state-replication protocol

SPs implement the FT-FW state-replication protocol to communicate each other. The main features of this protocol are:

- Extensibility: we can easily extend the replication message format to support new stateful firewall features without breaking backward compatibility. We detail this aspect, and the Synchronization message format in general, in Subsect. 5.1.
- Resilient replication under message delivery failures: our protocol exploits what we have defined in Sect. 3 as the *flow-state independency* property. Based on this property, the backup stateful firewall does not require the sequence of former flow-states $S_1 \rightarrow S_2 \rightarrow ... \rightarrow S_{k-1}$ to reach the consistent state S_k for a given flow F_i. This property allows our protocol to reduce the number of retransmissions under message reordering and omission situations since one single replication message containing the current flow-state S_k suffices to recover the flow F_i. We develop this idea in Subsect. 5.2.
- Message batching: the protocol allows to put several replication messages into one single network packet. Thus, we avoid sending small packets containing flow-state changes to improve network efficiency. Moreover, this reduces CPU consumption as the number of *send()* system calls is smaller. We cover this feature in Subsect. 5.3.

5.1. Replication message format

The replication messages are aligned to 32 bits and they contain data that is expressed in network-byte order (big endianness). The replication message always starts by a fixed header of 8 bytes. We have represented this header in Figure 3.

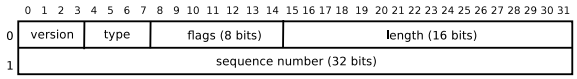

Figure 3: Synchronization message header

This header contains the following fields:

- Type (4 bits): This field indicates the type of the message. There are four types: Control, that is reserved for control messages. New, Update and Delete that contain flow-state data.
- Version (4 bits): That is set to zero, that is the first version of the replication protocol. We reserve this field in case that we need to define a new version of the replication protocol that need to break backward compatibility.
- Flags (8 bits): The protocol supports up to seven flags by now. One flag is reserved for future use. The existing flags are: a) *Resync*, that is used to request a full resynchonization with another stateful firewall node, b) *Ack*, that tells that this message contains a positive selective acknowledgment, c) *Nack*, that tells that this message contains a negative selective acknowledgment, d) *Alive*, to inform that this stateful firewall is alive but no flow-state changes have occured after some time, e) *Hello*, to reset message sequence tracking, f) *Hello back*, to confirm that the message sequence tracking has been reset; and g) *Out-of-sync*, to inform that it sends messages before the expected sequence number.
- Length (16 bits): The size of the replication message including this header.
- Sequence number (32 bits): The sequence number that is used to track message duplication and loss.

After this header, it follows the payload that contains the flow-state information. The payload contains a set of attributes that are expressed in Type-Length-Value (TLV) format. TLVs are used in many protocols due to their extensible nature, e.g. IPv4 options field and IPv6 extension headers.

The attributes are composed of a header:

- Type (16 bits): the attribute type according to the set of available stateful firewall features. Thus, the maximum number of attributes is limited to 65536.
- Length (16 bits): size in bytes of the attribute. This includes this header plus the payload size of this attribute without alignment to 32 bits.
- Value: this field is variable in size but it is always aligned to 32 bits.

In Figure. 4 we have represented a hypothetical payload in TLV format composed on two attributes, one of 32 bits and another of 64 bits.

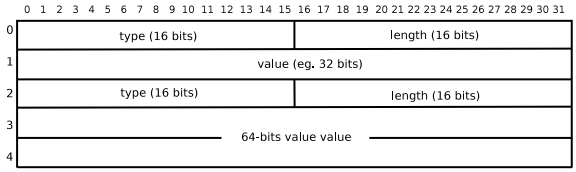

Figure 4: An example of a hypothetical payload in TLV format

The reasons why we have decided to use the TLV format to represent the flow-state variables, instead of the traditional fixed-layout format, are that:

1. Depending on the state transition, the amount of variables which have changed their value may vary widely. Thus, we only propagate the value of variables that have changed.
2. The number of variables that are part of a state also depends on the features enabled in the stateful firewalls. As some of these features may be present or not, we do not increase the size of the replication message with unused fields.
3. Since new features may introduce new variables, the TLV format guarantees backward compatibility. Thus, if we have a cluster composed of two stateful firewall Fw_x and Fw_y where Fw_y only implements a subset of features of Fw_x, the variables that represent such features would be ignored by Fw_y without breaking the protocol between firewalls. Nevertheless, we recommend that both stateful firewalls implement the same set of features.

In general, as opposed to fixed structure message format, the TLV format does not require to bump the version field of the protocol for every new feature not to break backward compatibility. Instead, you only have to add a new attribute and update your SP in case that you want to support this new feature. As said, old SPs ignore new attributes in the message as they do not know how to interpret them.

5.2. Replication protocol description

Our replication protocol is based on an incremental sequence number algorithm to ensure reliable flow-state replication. SPs that support our protocol:

a) do not wait for selective positive acknowledgments to send new messages. Thus, their behaviour is completely asynchronous.

a) do not discard messages whose sequence number are after the last sequence number that they have received.

This behaviour is based on the *flow-state independency* property that we have define in the system model (Sect. 3), that assumes that:

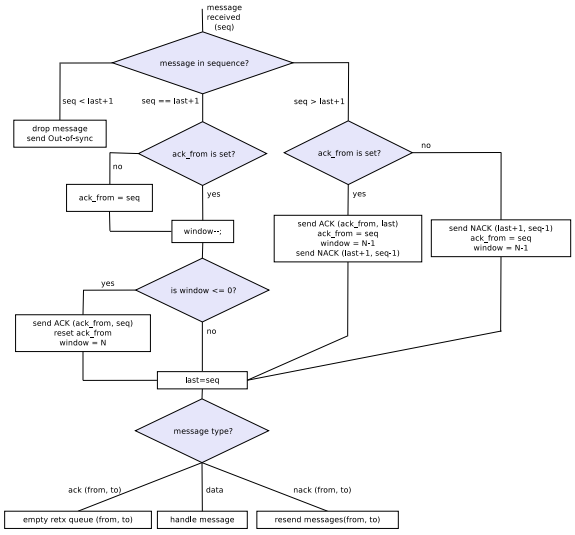

Figure 5: Sequence tracking decision diagram for input data messages

1. there is no dependency between flow-states that talk about different flows.
2. we can easily identify more recent flow-state changes because they are always included in messages whose sequence number are higher than others that contain previous flow-state changes.

We have represented the handling logic for input messages in Fig. 5. Initially, our protocol checks if the message sequence number is before, in or after what the SP has previously received. Depending on that, the protocol performs different actions, that are:

- *Before*: messages whose sequence tracking number is before to what we expect are discarded. They are likely to contain out-of-date flow-state information. In this case, we inform other SPs that they are out-of-sync to get their sequence number in sync by means of the *Hello* and *Hello Back* message flags.
- *In*: messages in the expected sequence are accepted and processed. If a set of messages are received in sequence, the SP replies with a positive selective acknowledgment once the window is filled.
- *After*: messages after the sequence number that the protocol expects are also accepted and processed. Basically, the protocol confirms with a positive selective acknowledgment what it has previously received. Moreover, it also sends a negative selective acknowledgment to inform that a set of messages have not been appropriately received.

The protocol allows two type of messages that are control and flow-state data messages. In the particular case of control messages, what the protocol does depends on the message type:

- *Positive Selective Acknoledgments* (*Ack*): this message contains a range of messages that have been successfully received by others. This implies that these messages can be removed from the retransmission queue.
- *Negative Selective Acknoledgments* (*Nack*): this message contains a range of messages that were not successfully received by the other stateful firewall node. This implies the complete or partial retransmission of the messages that have not been delivered.
- *Out of Sync*: this message informs that one SP is sending messages that are before the expected sequence number.
- *Hello*: that is a request to reset the current sequence number of one SP. The sequence number of this message plus one is used to set the new current expected sequence number.

- *Hello Back*: that confirms that this SP has reset its expected sequence number.
- *Alive*: this message is sent if no updates occur after a configurable amount of time. This control message is also used to notice message loss since *Alive* messages after the current sequence number trigger the send of an *Ack* plus *Nack*. In our experiments (Sect. 7), we have set that the time between two *Alive* messages is 1 second. A lower value increases the amount of *Alive* messages but it will improve the resiliency of the protocol under message loss scenarios.

With regard to flow-state data messages, they contain one flow-state change information that talk about the flow-state creation, update or deletion in TLV format as we have described in this section.

5.3. Batching

We propose a batching mechanism that improve network efficiency by avoiding the transmission of small packets. This mechanism consists of stacking several messages into one single packet. We assume that the maximum size of the packet is equal to the *Maximum Transfer Unit* (MTU) of the Ethernet interface. The batching mechanism is particularly useful in three cases:

1. *Resynchronization*: one stateful firewall FW_x requests the whole set of flow-states stored in another stateful firewall FW_y.
2. *Retransmission*: one or several messages are omitted due to failures.
3. *High rate of flow-state changes*: several flow-state changes that occur at the same time.

In these situations, the batching allows to reduce the number of transmitted packets.

6. Discussion

This section discusses design decisions that we have faced while architecting and implementing our proposed approach. We compare them to other existing solutions, such as OpenBSD PFSync in the FOSS community; and Cisco PIX/ASA and Checkpoint Firewall-1 in the closed-source side. For the latter cases, we refer to the limited internal implementation details that vendors provide in user manuals and existing books.

6.1. Synchronous vs. Asynchronous protocol

In the particular case of stateful firewalls, synchronous flow-state replication means that each packet would have to wait until one flow-state change is propagated to the backup stateful firewalls. Then, once all the backup firewalls have confirmed that the flow-state change has been correctly received, the packet continues its journey.

This approach ensures that all backup firewalls are one-copy equivalences. However, it increases latency and reduces bandwidth throughput since packets have to wait until flow-states are successfully propagated. Thus, we ensure that all the flows are successfully recovered at the cost of dramatically reducing network performance.

The performance harm that synchronous replication incurs renders it not feasible for our purposes. For that reason, we have to consider asynchronous flow-state propagation. However, asynchronous replication implies that packets leave without having guaranteed that flow-state changes have been correctly propagated. This can lead to the following situation:

1. A packet triggers a flow-state change for F_i. Thus, $S_n \rightarrow S_{n+1}$ happens for F_i.
2. The flow-state changes $S_n \rightarrow S_{n+1}$ is sent to other SPs. However, a failure occurs before the SP successfully propagates the flow-state change through the dedicated link to others.
3. The backup stateful firewalls remain in state S_n but the flow F_i is in state S_{n+1} in the primary firewall. Thus, the backup is out-of-date and, therefore, would not be able to recover the flow F_i appropriately.

As we will show in the experiments section (Sect. 7), this is specifically a problem for small flows, ie. those that start and finish almost inmediately. Thus, the asynchronous approach does not ensure that short flow-states are successfully propagated in time. Therefore, these flows may not be appropriately recovered under failures as we will show.

To the knowledge of the authors, all existing solutions in the FOSS and closed-source communities use asynchronous flow-state replication.

6.2. Kernel-space versus User-space implementation

Modern monolitic operating systems like GNU/Linux and OpenBSD implement stateful firewalling in kernel-space for performance reasons. In the particular case of OpenBSD PFSync (McBride, 2004), state-replication is also implemented in kernel-space.

We consider that implementing state-replication in kernel-space is problematic. Modularity and fault-isolation are important aspects to achieve fault-tolerance (Herder et al., 2006). The encapsulation of the different components that interact with each other is a good property to avoid that faulting components affect sane ones. Moreover, reducing the amount of lines of code that are executed in privileged mode intuitively reduces the chances to crash the system due to software programming errors.

Therefore, if the state-replication implementation runs in kernel-space, one bug in it may lead to the crash of the firewall. In our opinion, this seems contradictory, since the goal of state-replication is to improve the overall availability. If the state-replication solution hits one bug, we should be able to re-launch it safely or simply disable it temporarily until some bugfix for the problem is released.

For these reasons, we consider that implementing state-replication in user-space is the correct approach to increase reliability. However, in order to implement the state-replication in user-space, you require some interface to communicate your stateful firewalling subsystem in kernel-space with the state-replication software that runs in user-space. In our case, we decided to implement one Netlink interface (Pablo Neira-Ayuso, 2010) for the Linux stateful firewalling subsystem. This interface provides a flexible and extensible messaging system to communicate between kernel and user-space. This idea has been extracted from the design of micro-kernel operating systems.

But not all are advantages. This design decision has some impact in terms of performance since it implies more CPU consumption, as more code needs to be executed to build, to deliver and to parse messages between kernel and user-space. We pay attention to this performance penalty in Sect. 7.

Unfortunately, we have no information regarding where Cisco and Checkpoint implement state-replication.

6.3. Reliable versus Unreliable state-replication

Networks produce errors which may happen in the sender, in an intermediary equipment or in the receiver. Generally speaking, there are several errors that may occur to the state-change messages that are transmitted, they are: omission, duplication, re-ordering and corruption. Since the software that implements the state-replication is supposed to achieve fault-tolerance, it must work as its best under network errors. Thus, it must avoid state-change loss due to any of the previous errors.

If an unreliable unicast UDP or multicast is used to transfer state-changes, then network errors may result in out-of-synchronization backup firewalls. Thus, the backup firewalls may become useless to recover established flows of the primary firewall. Using some existing reliable protocol avoids this problem. However, some application-level state-replication protocol may exploit the semantics of the stateful firewalls to reduce the number of retransmitted messages under any of the failures that compromise message delivery. This is the case of our FT-FW protocol. There are also generic cluster-oriented transport-level protocols like TIPC (Maloy, 2006) that can be used to reliably propagate state-changes.

To the knowledge of the authors, OpenBSD PFSync uses an unreliable multicast state-replication protocol. Cisco ASA and Checkpoint Firewall-1 use multicast-based protocols. However, according to the existing literature, we do not know if they implement some reliability mechanisms.

6.4. Batching flow-state replication updates

As we detailed in 5.3, we provide advanced batching mechanisms to avoid the extra overhead that several consecutive small messages may incur. At the same time, we try to delay flow-state replication as less as possible. OpenBSD PFSync (Attebury and Ramamurthy, 2006) provides similar batching mechanisms. Other vendors apply more agressive batching schemes to roughly reduce this overhead. This is the case of Checkpoint Firewall-1 in which flow-states are delivered in incremental batches every 100ms (Welch-Abernathy, 2004). Regarding Cisco, the user manual available on the Internet (Cisco Systems Inc., 2011) vaguely states that their ASA 55xx solutions continually passes flow-state information.

6.5. Security considerations on the state-replication

The state-replication must be done using some measure of protection to avoid possible flow-state leakage that a third party may use to know what traffic the firewall filters. There are different approaches to solve this situation:

- Use one dedicated link. This is approach requires one spare port in one of the Network Interfaces (NIC) per firewall node. This may not be possible if all ports are in use. In that case, you have to acquire a new NIC with more ports to solve this issue. This implies some extra cost to upgrade your NIC though.
- Use encryption and authentication. We propose the use of IPSec in transport mode to encrypt and to authenticate all the state-replication traffic between two firewall nodes. As an alternative, we plan to implement our FT-FW protocol over the recent Datagram Transport Layer Security (DTLS) standard (Rescorla and Modadugu, 2006). There exists FOSS implementations in recent versions of the openSSL (The OpenSSL Project, 2011) and GNUTLS (Free Software Foundation, 2011) libraries. We started initial DTLS support for our implementation, however, we found that the existing codebase is not mature enough for our evaluation purposes. Nevertheless, we consider that DTLS is a promising option for the near future.

OpenBSD and Cisco allow to choose between the dedicated link and IPSec approaches to avoid eavesdropping of flow-states synchronization messages.

6.6. Software diversity and compatibility

Software and hardware diversity are good to achieve fault-tolerance (Avizienis and Kelly, 1984). Contrary to this principle, vendors like Cisco, Checkpoint and OpenBSD recommend using the same software versions and the same hardware for their failover solutions.

Achieving hardware diversity is relatively easy by acquiring systems from different vendors. However, software diversity requires that different implementations (from different developer teams) are able to interoperate each other. This is not possible so far. In order to make this possible, in the specific domain of Free/OpenSource solutions, we propose extending OpenBSD PFSync to support our proposed state-replication protocol. The ACL management should not be a problem if some abstract language to express the rule-set is used (Pozo et al., 2008).

Regarding compatibility, most vendors tend to break the state-replication protocol between versions (Gwynne, 2009). Thus, diversity in terms of software versions is not possible. This occurs because the format that they use for their state-replication protocol is based on fixed headers. Then, if you want to extend it, you have break the protocol layout. We avoid this problem by making extensive use of TLVs. These require more CPU resources to build and to parse messages. However, this makes easier to keep our protocol forward and backward compatible, as well as to make extensions to support new stateful firewalling features.

7. Experiments

The experiments that we have performed to evaluate our proposed solution depend on three network performance metrics:

1. *Session rate*, that determines the number of flows per second that the firewalls can filter. In order to obtain the maximum session rate that the solution can filter, we have to generate flows with no data transfers. Thus, the flows are set up and tear down almost inmediately. This is the worst case for state-replication since it generates the maximum number of flow-state changes per second.
2. *Data rate*, that talks about the amount of data per second that the solution can filter. This metric is expressed in bytes per second. To obtain the maximum data rate we have to generate flows that perform large data transfers. The maximum data rate provides the best case for state-replication since it generates the minimum number of flow-state changes per second.
3. *Round-Trip Time* (RTT), that is the time that a packet takes to go from the client to the server and back. The RTT is an important metric since our state-replication solution is completely asynchronous. Thus, packet forwarding and flow-state propagation may race. Intuitively, the chances to propagate a flow-state change timely improve if the RTT increases.

With regards to availability, we have used two metrics to evaluate it:

1. *Flow recovery*. We have used the statistics mode of recovered flows and their deviation. The failures have been triggered by unplugging the dedicated link, that is used to propagate flow-state changes, and one of the main links, that connects the firewalls with the HTTP client and server.
2. *Time to recover*. We have measured the time that flows take to recover (if they indeed successfully recover) after failures.

We have grouped the results in two subsections: network performance (Sect. 7.1) and availability (Sect. 7.2). Specifically, measuring availability is a daunting task since the measurements show great variability. For that reason, we have repeated the experiments up to five times to normalize the results.

Our environment consists of four HP Proliant 145g2 with one AMD dual core 2.2GHz using 1 GEthernet links in the Local Area Network (LAN). Two of the systems have been used as firewalls in a simple Primary-Backup setup; and the other two act as HTTP client and server respectively. We have repeated the experiments with both firewalls in a Multi-Primary workload sharing setup. We have also tuned the interrupt handling affinity of each CPU core. Thus, the network card's interrupts in the firewalls are always handled by the same CPU core to improve performance.

We have selected Linux as operating system and the *iptables* (Netfilter Core Team, 1999) firewalling tool which are widely used these days and because of their source code availability. We have used our implementation of the SP, that is distributed within the *conntrack-tools* (Neira-Ayuso, 2006a) package (version 1.0.0), which is the de-facto solution for GNU/Linux firewalls nowadays[4], to evaluate state-replication. This package contains the userspace daemon *conntrackd* that propagates flow-states asynchronously between stateful firewalls. At the time this article was written, our daemon supports FT-FW, TCP and UDP as replication protocols. We have selected *keepalived* (Cassen, 2002), a high-availability manager that implements VRRP (Hinden, 2004.), as FDM. This software also provides extended multilayer availability monitoring capabilities and it supports fail-stop scenarios.

We have used our own HTTP client traffic generator (Neira-Ayuso, 2008) and the *httpterm* (Willy Tarreau, 2006), that is a simple HTTP benchmarking server. Basically, the HTTP client starts up to 200 TCP sessions with the server and they retrieve one object of variable size per flow. The firewalls perform stateful filtering and Network Address Translation (NAT). We have also injected errors in the dedicated link to evaluate the resiliency of the FT-FW replication protocol by means of *netem* (Hemminger, 2005).

7.1. Network performance

In Fig. 6 we have represented the performance results that we have obtained in a simple Primary-Backup setup depending on the object size. With state-replication enabled, our solution reduces the amount of session/s that our firewall can handle for objects smaller than 50KB. The amount of resources invested in the state-replication slightly reduces the session and data rate. Specifically, there is a peak of 14% of session rate drop aprox. for objects between 0 and 1KB. This figure also includes the performance evaluation of state-replication over IPSec. Basically, you can notice a reduction in terms of session/s of 6-9% for objects smaller than 5KB.

We repeated the experiments with an emulation of 50ms RTT by means of *netem* (Hemminger, 2005). Thus, the packets that travel from the client to the server and back take 50ms. We have selected this amount of time since this is the common RTT in communications between clients and servers in the Internet via widespread DSL links. The results in Fig. 7 shows that performance, in terms of session/s, is reduced since the clients take longer to make the HTTP hit. Data rate is not affected as expected. We have observed negligible difference in terms of network performance with and without flow-state replication enabled.

We have also repeated the experiments in the Multi-primary load-sharing setup. The results in Fig. 8 show that, without state-replication, we obtained around 45000 sessions/s, which is approximately double of what we have obtained in the Primary-Backup setup (see Fig. 6). This occurs because both firewall are actively filtering traffic, instead of having one idle backup like in the Primary-Backup setup. With state-replication enabled, there is a peak performance drop of around 30%. We obtain the same results with and without state-replication for 5KB objects. No significant changes are noticed with regards to the network throughput in this scenario.

[4]Several firewall vendors reported us that they are using our implementation of the FT-FW architecture in commercial solutions, they are Vyatta, 6WIND, Edenwall and Astaro AG; free software solutions like *Firewall Builder 4* also integrates our implementation.

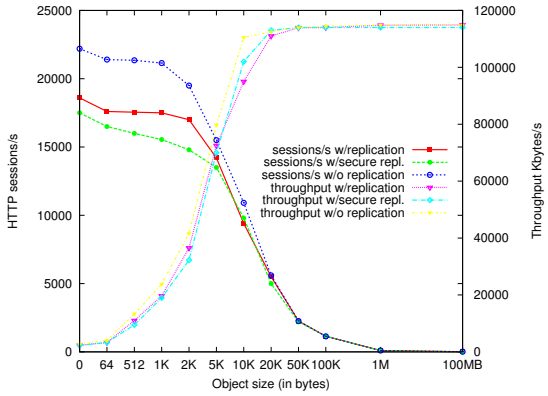

Figure 6: Performance evaluation with negligible RTT

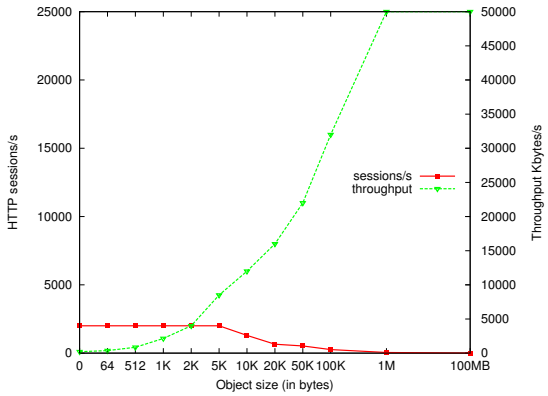

Figure 7: Performance evaluation with 50ms RTT

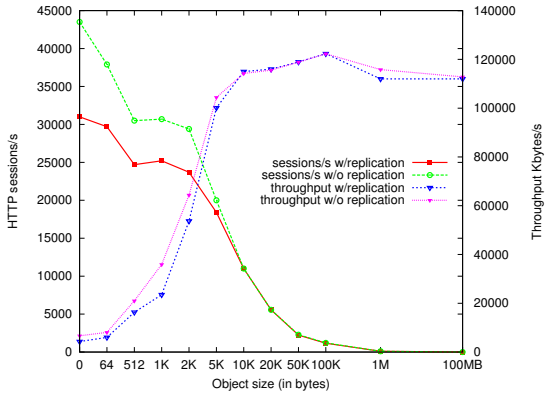

Figure 8: Performance in the Multi-primary load-sharing setup

7.2. Availability

In Fig. 9, we have represented the recovery results. We have assumed negligible RTT (as it occurs in a LAN setup). The results show that the state-replication improves the recovery for objects bigger than 5KB. Note that, for objects smaller than 5KB, the recovery is not improved. Thus, it is similar to having no flow-state propagation at all but at the cost of reducing performance (as we mentioned in Fig. 7.1). We do not include recovery evaluation results of state-replication over IPSec since we did not notice any significant variation regarding clear flow-state synchronization based on the use of one dedicated link.

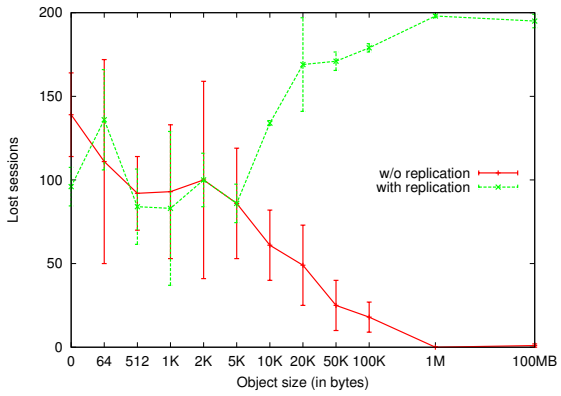

Figure 9: Recovery with negligible RTT

We have repeated the experiments by emulating 50ms RTT between the client and the server. The results in Fig. 10 show that the recovery is improved. Basically, non-negligible RTT reduces the session rate since the HTTP traffic generator tools require more time to perform the 200 client requests. Thus, the amount of flow-state changes is smaller. We can conclude from this that usual RTT of Wide Area Networks (WAN) improves the chances to deliver flow-state changes timely. This is particularly interesting if the firewalls are deployed in the LAN or the WAN.

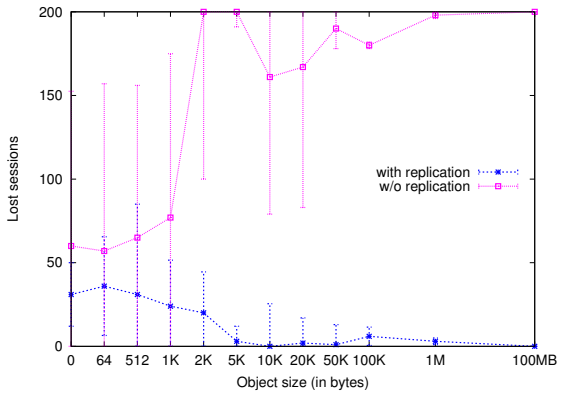

Figure 10: Recovery with 50ms RTT

In order to evaluate the resiliency of our FT-FW protocol, we have emulated a packet loss of 20% in the dedicated link[5] with netem (Hemminger, 2005) and we have compared it with UDP and TCP as replication protocols. These protocols have the following properties:

[5]As the results show, this packet loss rate is sufficient to justify that TCP and UDP cannot compete with our FT-FW protocol even under low packet loss rate.

- UDP is unreliable. Thus, flow-state propagation is also unreliable. With UDP, packet loss results in flow-state change information that is not ever successfully delivered to the backup firewall. This reduces the chances that the backup has to recover flows under failures.
- TCP ensures reliable state-replication. However, TCP behaviour under packet loss does not ensure that flow-state changes are delivered timely. Basically, TCP in-order delivery results in a slow progression in the flow-state propagation, since every packet is enforced to be delivered even if it contains obsolete flow-state information. Intuitively, this also may lead to reduce the chances to recover flows under failures.
- FT-FW ensure reliable state-replication by exploiting the stateful firewall semantics. Basically, it only resends the current flow-state under failures, not the entire sequence of flow-state changes that is not required to reach the current flow-state. This protocol aims to improve recovery under packet loss situation in the dedicated link.

The results in Fig. 11 show that FT-FW behaves better under failures as expected. Specifically, for objects larger than 10 KB, TCP and UDP do not provide good results. As far as we know, most fault-tolerant solutions for stateful firewalls vendors provide nowadays are based on TCP and UDP protocols. That is not a good choice according to the results that we have obtained in this work.

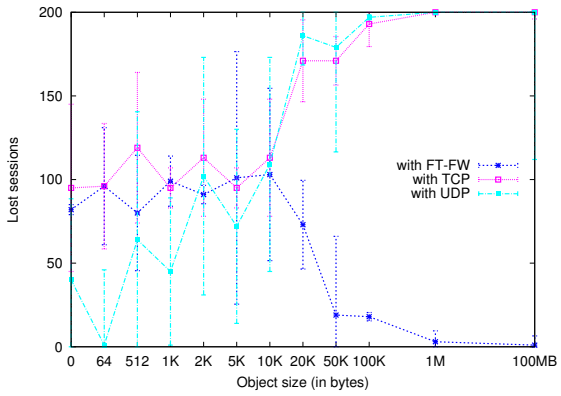

Figure 11: Recovery with 20% of packet loss in dedicated link

We have expressed the experimental results in the Multi-primary load-sharing in Fig. 12. The results show that the state-replication slightly improves the recovery for objects smaller than 5KB. The real improvement comes for bigger objects.

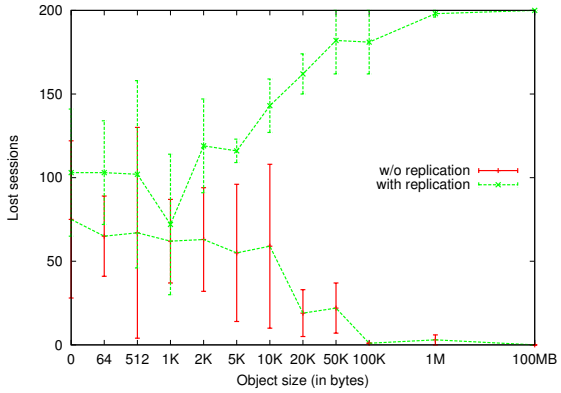

Figure 12: Recovery with Multi-primary load-sharing

7.2.1. Time to recover

The time to recover greatly depends on the FDM since it basically decides when to migrate to the backup firewall. Since our selected FDM is a VRRP-based implementation, it depends on a parameter that is specified in the VRRP specification that is the advertisement interval. Basically, VRRP sets the failover time after three times the advertisement interval. We have used an advertisement interval of 2 seconds. Thus, the recovery occurs 6 seconds after the failure has been detected in our setup.

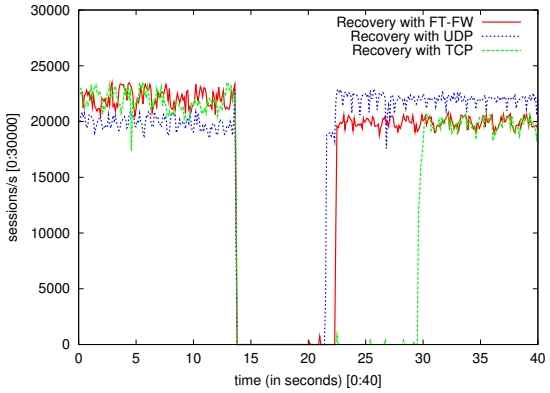

Figure 13: Time to recover

In Fig. 13, we have represented the recovery time depending on the state-replication protocols. Basically, our experiments show that, with no errors in the dedicated link, the UDP and FT-FW protocols recover after 6-7 seconds, which is negligible if we consider that VRRP takes 6 seconds to trigger the failover. TCP provides the worst results since the recovery occurs 15 seconds after the failure. Note that TCP provides the worst results due to the strict in-order delivery and the congestion control.

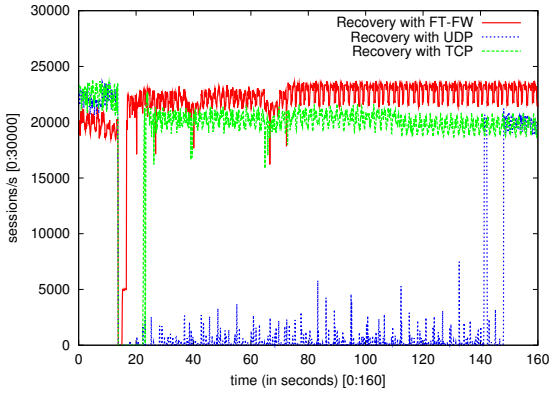

Figure 14: Recovery time under 20% packet loss

We have repeated the experiments emulating a packet loss of 20% in the dedicated link. We have represented the results in Fig. 14. We can note that the successful recovery with UDP takes around 130 seconds. This occurs because the missing flow-state information confuses the stateful filtering. Thus, the new primary firewall takes long to reach a consistent state. The best results are provided by our FT-FW protocol, that takes around 9 seconds. It follows TCP that takes around 15 seconds, that is similar to the previous results without failures.

8. Real-world experiments

We have deployed our solution in the network of the Departament of Computer Languages and Systems of the University of Sevilla. Basically, this network is composed of three segments:

- Internal DMZ, which contains Windows servers that can be accessed via Remote Desktop and SMB; and shared printers.
- External DMZ, where the HTTP/S server that hosts the website of the departament resides as well as several servers running virtualization solutions that are used to run experiments remotely from both home and the department offices.
- Client network, where you can find around 70 desktop computers that are used by the teachers, administrative and technical staff.

We have deployed one primary-backup cluster that is composed of two firewalls. The primary firewall owns 24 public addresses for servers that are located in the external DMZ. Traffic going to these servers is managed with destination NAT rules. On the other hand, traffic leaving the external DMZ and the client network is source-NATted. The ACL is composed of 500 rules approximately.

The hardware in use as firewalls is the following:

- Sun Fire X4100, two dual-core AMD Opteron processors, 4 Gbytes RAM.
- Sun Fire X4150, two quad-core AMD Opteron processors, 32 GBytes RAM. memory.

Initially, the network was protected by one single firewall. Later on, the department acquired one extra firewall to deploy the failover setup. This is not a problem since you do not need exact twin hardware for this purpose.

Both firewalls run Debian Linux 6.0 with kernel 2.6.32, our implementation available in the *conntrack-tools* 1.0.0 (Neira-Ayuso, 2006a) and Keepalived 1.1.20 (Cassen, 2002) as HA manager. We selected the older equipment to run our tests to cover the worst case. We have represented the performance results of one day of activity in Fig. 15.

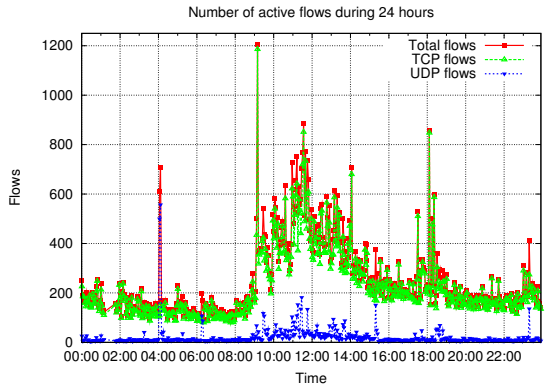

Figure 15: Performance measurements in the primary firewall during 24 hours

With state-replication disabled, *sysstat* (Sebastien Godard, 2011) reported the following CPU consumptions: 0.16% time invested in executing user-space code, 0.06% in kernel-space code, 0.17% waiting for the disk to write firewall logs and remaining 99.60% idle. Even during the peak hour at 9am, the results that we have obtained are similar.

We noticed few impact in terms of CPU consumption after enabling state-replication. Basically, *sysstat* reported some extra CPU consumption: 0.25% time invested in executing user-space code, 0.12% in kernel-space code, 0.17% waiting for the disk to write firewall logs and remaining 99.46% idle.

For further evaluation, we started 200 file downloads via HTTP on the Internet from the client network. Then, we disconnected the cable of the internal DMZ to trigger the failover. As a result, we noticed no disruption in any of the on-going downloads.

9. Conclusions and future work

In this work, we summarize the FT-FW architecture, we provide an improved and more detailed version of our replication protocol and we enhance our previous experimental evaluations based on our implementation, that is released under a FOSS license. We also discuss several design decisions comparing them to those of existing solutions.

We plan to extend this work with proactive or semi-proactive fault-tolerance to avoid the extra cost (in terms of resources) of our reactive flow-state propagation. In order to do that, we are currently studying hybrid solutions that are composed of hardware-based failure-detection facilities and simple online machine learning mechanisms to provide more advanced fault-tolerant solutions for stateful firewalls. We already made some research in this direction (Neira-Ayuso et al., 2008b) that we want to extend.

We are also working towards the implementation of our FT-FW protocol over Datagram Transport Layer Security (DTLS) (Rescorla and Modadugu, 2006) as a lightweight replacement of IPSec to secure the flow state-replication.

We plan to update Daniel Hartmeier's article that compares OpenBSD versus Linux as firewalls (Hartmeier, 2001), which is now more than ten years old. Basically, we will compare OpenBSD/PF/PFSync versus Linux/iptables/conntrack-tools. These require the comparison of OpenBSD and Linux as routers, with no firewalling at all, then we plan to continue enabling stateless firewalling, stateful firewalling and the failover setup, both the primary-backup and multi-primary setups.

Acknowledgments

This work has been partially supported by the Spanish Ministerio de Ciencia e Innovación through a research project (grant TIN2009-13714) and Feder (ERDF). We would like to thank reviewers of Computer Network and Security journal for their useful suggestions and ideas.

References

Al-Shaer, E., Hamed, H., 2006. Taxonomy of Conflicts in Network Security Policies. IEEE Communications Magazine 44 (3).

Al-Shaer, E., Hamed, H., Boutaba, R., Hasan, M., 2005. Conflict Classification and Analysis of Distributed Firewall Policies. IEEE Journal on Selected Areas in Communications (JSAC) 23 (10).

Attebury, G., Ramamurthy, B., 2006. Router and firewall redundancy with openbsd and carp. In: bf ICC '06. IEEE International Conference on Communications. Vol. 1. pp. 146–151.

Avizienis, A., Kelly, J., 1984. Fault tolerance by design diversity: Concepts and experiments. Computer 17, 67–80.

Ayari, N., Barbaron, D., Lefevre, L., Primet, P., 2007. T2CP-AR: A system for Transparent TCP Active Replication. In: AINA '07: Proceedings of the 21st International Conference on Advanced Networking and Applications. pp. 648–655.

Ayari, N., Barbaron, D., Lefèvre, L., Vicat-Blanc Primet, P., Jul. 2008. Fault tolerance for highly available internet services: Concepts, approaches, and issues. IEEE Communications Surveys and Tutorials 10 (2), 34–46.

Bartal, Y., Mayer, A., Nissim, K., Wool, A., Nov. 2004. Firmato: A novel firewall management toolkit. ACM Transactions on Computer Systems 22 (4), 381–420.

Cassen, A., 2002. Keepalived: Health checking for LVS & high availability.
URL `http://www.linuxvirtualserver.org`

Cisco Systems Inc., 2011. Cisco asa 5500 series configuration guide: Information about high availability. http://www.cisco.com/en/US/docs/security/asa/asa82/configuration/guide/ha_overview.html.

Free Software Foundation, 2011. GnuTLS - GNU Transport Layer Security Library.
URL `http://www.gnutls.org`

Gorti, A., Oct. 2006. A faul tolerant VoIP implementation based on Open Standards. In: IEEE Proceedings EDCC-6. pp. 35–38.

Gouda, M., Liu, A., Jun. 2005. A model of stateful firewalls and its properties. Proceedings of the International Conference on Dependable Systems and Networks (DSN), 128–137.

Gwynne, D., 2009. Redesign of the pfsync protocol.
URL `http://www.undeadly.org/cgi?action=article&sid=20090220014805`

Hartmeier, D., 2001. http://www.benzedrine.cx/pf-paper.html. http://www.benzedrine.cx/pf-paper.html.

Hemminger, S., 2005. Network Emulation with NetEm. Linux Conf Australia.

Herder, J. N., Bos, H., Gras, B., Homburg, P., Tanenbaum, A. S., 2006. Construction of a highly dependable operating system. In: In Proc. 6th European Dependable Computing Conf.

Hinden, R., Apr. 2004. RFC 3768: Virtual Router Redundancy Protocol (VRRP).

Maloy, J., May 2006. TIPC: Transparent Inter Protocol Communication protocol.

Marwah, M., Mishra, S., Fetzer, C., Jun. 2003. TCP server fault tolerance using connection migration to a backup server. In Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN), 373–382.

McBride, R., 2004. Pfsync: Firewall Failover with pfsync and CARP.
URL `http://www.countersiege.com/doc/pfsync-carp/`

Nagarajan, A. B., Mueller, F., Engelmann, C., Scott, S. L., 2007. Proactive fault tolerance for hpc with xen virtualization. In: ICS '07: Proceedings of the 21st annual international conference on Supercomputing. ACM, New York, NY, USA, pp. 23–32.

Narasimhan, P., Dumitras, T. A., Paulos, A. M., Pertet, S. M., Reverte, C. F., Slember, J. G., Srivastava, D., 2005. MEAD: support for Real-Time Fault-Tolerant CORBA: Research Articles. Concurr. Comput. : Pract. Exper. 17 (12), 1527–1545.

Neira-Ayuso, P., 2006a. Conntrack-tools: The Connection Tracking User-space Tools for GNU/Linux.
URL `http://conntrack-tools.netfilter.org`

Neira-Ayuso, P., 2006b. Netfilter's Connection Tracking System. In :LOGIN;, The USENIX magazine 32 (3), 34–39.

Neira-Ayuso, P., 2008. http-client-benchmark: a simple HTTP client benchmarking tool similar to Willy Tarreau's inject.
URL `http://1984.lsi.us.es/git/http-client-benchmark`

Neira-Ayuso, P., Gasca, R. M., Lefevre, L., Feb. 2008a. 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'08). pp. 573–580.

Neira-Ayuso, P., Gasca, R. M., Lefevre, L., 2009. Demystifying cluster-based fault-tolerant firewalls. IEEE Internet Computing 13, 31–38.

Neira-Ayuso, P., Lefevre, L., Gasca, R. M., 2008b. hft-fw: Hybrid fault-tolerance for cluster-based stateful firewalls. Parallel and Distributed Systems, International Conference on 0, 525–532.

Neira-Ayuso, P., Lefevre, L., Gasca, R. M., Oct. 2008c. 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.

Netfilter Core Team, 1999. iptables: The free software firewalling tool for linux. http://iptables.netfilter.org.

Pablo Neira-Ayuso, Rafael M. Gasca, L. L., Aug. 2010. Communicating between the kernel and user-space in Linux using Netlink sockets. Software: Practice and Experience 40 (9).

Powell, D., Feb. 1994. Distributed Fault Tolerance: Lessons learnt from Delta-4. In: IEEE Micro. Vol. 14. pp. 36–47.

Pozo, S., Ceballos, R., Gasca, R. M., 2008. Afpl, an abstract language model for firewall acls. In: ICCSA (2)'08. pp. 468–483.

Pozo, S., Ceballos, R., Gasca, R. M., 2010. A Quadratic, Complete, and Minimal Consistency Diagnosis Process for Firewall ACLs. Advanced Information Networking and Applications (AINA).

Rescorla, E., Modadugu, N., 2006. RFC 4347 - Datagram Transport Layer Security.
URL `http://tools.ietf.org/html/rfc4347`

Robertson, A., 1999. Linux HA project.
URL `http://www.linux-ha.org`

Sebastien Godard, 2011. System performance tools for linux. http://sebastien.godard.pagesperso-orange.fr/.

Sultan, F., Bohra, A., Smaldone, S., Pan, Y., Gallard, P., Neamtiu, I., Iftode, L., Apr. 2005. Recovering Internet Service Sessions from Operating System Failures. In: IEEE Internet Computing.

Sultan, F., Srinivasan, K., Iyer, D., Iftode, L., 2002. Migratory TCP: Connection Migration for Service Continuity in the Internet. In: 22nd International Conference on Distributed Computing Systems.

The OpenSSL Project, 2011. OpenSSL - Open Source library that implements the SSL and TLS protocols.
URL `http://www.openssl.org`

Vallee, G., Charoenpornwattana, K., Mar. 2008. A framework for proactive fault tolerance. In: Proceedings of 3rd international conference on Availability, Reliability and Security (ARES).

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

Vishwanath, K. V., Nagappan, N., 2010. Characterizing cloud computing hardware reliability. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11. pp. 193–204.

Welch-Abernathy, D. D., 2004. Essential Check Point FireWall-1 NG: An Installation, Configuration, and Troubleshooting Guide. Pearson Higher Education.

Welte, H., 2004. ct_sync: state replication of ip_conntrack. Linux Symposium, Ottawa, Canada 2.

Willy Tarreau, 2006. httpterm: HTTP server benchmarking tool.
URL `http://1wt.eu/tools/httpterm/`

Wool, A., 2004. A quantitative study of firewall configuration errors. IEEE Computer 37 (6), 62–67.

Zhang, R., Adelzaher, T., Stankovic, J., Mar. 2004. Efficient TCP Connection Failover in Web Server Cluster. In: IEEE INFOCOM 2004.