Flow Digest: A State Replication Scheme for Stateful High Availability Cluster

Yi-Hsuan Feng+  Nen-Fu Huang*+, Rong-Tai Liu*  Meng-Huan Wu*
+Dep. of Computer Science  *Inst. of Communication Engineering  # Broadweb Corp.
National Tsing-Hua University  National Tsing-Hua University  Hsin-Chu, Taiwan
Hsin-Chu, Taiwan

Abstract—Stateful tracking is a popular technique in firewall filtering and state replication is used to provide reliable connectivity. This paper proposes a new approach for improving existing state replication protocols which ensure state consistency amongst the nodes of a stateful HA cluster. Our goal is to develop a new scheme which reduces the update overhead in the face of both low and high connection loads in order to maximize the capacity and scalability of a high availability cluster. A new representation called flow digest is proposed. Also the ways to use flow digest structures to update state changes, recover the connections after a failover, and solve the state inconsistency are presented. The main advantage of the proposed method is to reduce the bandwidth consumption on state replication. The simulation results show that the proposed scheme reduces the number of update messages and, more importantly, eliminates typically at least 86% of bandwidth consumption compared to current solutions.

Keywords—Firewall; High Availability; Stateful Tracking; State Replication; Scalability

I. INTRODUCTION

The vulnerability of the Internet to malicious traffic promotes the widespread deployment of security systems such as firewall, content filtering, anti-virus gateway and intrusion prevention system (IPS) in different scale networks; from hosting companies to ISPs. One essential mechanism to provide robust security of these systems is to derive state information and update it dynamically from currently active TCP connections by a state table. By this so-called stateful tracking, any non-SYN packet which does not have an entry in the state table will be dropped and a legitimate connection is monitored continuously to ensure its compliance with the TCP specification. A filtering component (e.g., DDoS prevention) may also make its control decision on an arriving packet depending on its cumulative state information. Oppositely, a basic stateless inspection solely evaluates every individual packet without the assistance of historic data. A detailed analysis of firewall performance can be found in [1] for stateful and stateless filtering with different ruleset sizes.

On the other hand, if the networking is guaranteed for 24x7 running time, the security systems must have the long-term durability as well. This is achieved by adopting an adequate high availability (HA) strategy to ensure there are redundancy for critical networking components and fundamental data duplication in the environment, as well as a sophisticated failover mechanism for systems and data. The HA is implemented by configuring two or more facilities as a single HA cluster and to the internal/external network this cluster appears to operate as a single node and provides the service in a consistent way at any given time. Consider Fig. 1 where a cluster operates in the active-passive mode composed of one primary node, one slave node, and a dedicated crossover link for isolating the traffic among them. The network-level redundancy [2],[4],[5],[8] ensures the availability by link-level and equipment-level backup. In detail, all nodes send the keep-alive messages periodically for failure detection. If the primary is recognized to stop processing network traffic, a slave in hot standby automatically becomes the new primary through an election and takes over the works of old primary. After failover process, the cluster continues to handle the network traffic without interruption.

In a stateful HA scenario, a network-level redundancy is not satisfactory and a state replication protocol (SRP) is necessary; the state information of existing connections is so vital and must be sent from a primary to slaves as well to provide reliable connectivity and continuous state tracking when the primary goes down. Without state replication, all active connections will be blocked by the new primary due to its empty state table. The goal of state replication is to ensure the state information at a slave reflects what connections are active in the primary state table.

The OpenBSD and Linux, two widely used operating systems in open source, designed their SRPs to provide inexpensive, high available firewall filtering. After the release 3.5, OpenBSD uses the component pfsync [7] to replicate the state information of IP Filter in conjunction with CARP.
addition, a combination of ct_sync [6] and keepalived gives a solution to support stateful HA for Linux netfilter subsystem [9]. As illustrated in Fig. 1, it is important to note that both pf-sync and ct_sync protocols rely on the update messages containing the connection information to synchronize all state insertions, changes (e.g., from the SYN-SENT state to EST state), and deletions among the primary and other backup nodes via multicasting on specific interfaces. The messages are handled by a slave immediately to merge state modifications to the local state table. In addition, the update timing is also critical to the system performance. These messages are sent in a scheduled manner or when the delayed output queue is fulfilled. For example, in ct_sync, the queue size is 32 by default.

In this paper, we address the issue of state replication for a stateful cluster which processes the pass-through flows between the internal and external networks. We first discuss the potential overhead of existing SRPs in section 2. In section 3, we propose a new scheme for state replication to reduce the update cost. The different phases of our scheme, the update criteria, and the mechanisms to solve the state inconsistency are also discussed in this section. The evaluations on estimated overheads by simulations are given in section 4 and the results indicate the proposed scheme with much smaller bandwidth consumption is more scalable than existing solutions. Section 5 and section 6 give a discussion and conclusions of the proposed method.

Some items and assumptions are defined as follows. First, a TCP connection is identified by a flowID, i.e., four-tuple <DstIP, SrcIP, DstPort, SrcPort>, and a properly initialized connection (i.e., through a valid 3-way handshake) handled by the primary is considered as an active entry in the state table. Second, every node in the same cluster uses the identical ruleset and system configuration. Third, since the TCP traffic is the protocol to dominate a majority of the Internet traffic, we focus on the TCP state table and state consistency in the cluster. Furthermore, when we mention “current SRPs” in the paper, we are referring to the pf_sync and ct_sync and an update message of current SRPs mainly contains a flowID. Finally, there are two sub-varieties of HA strategies: active-backup mode for redundancy and active-active mode for load balancing. Throughout this paper we restrict attention to the backup mode for redundancy and active-active mode for load balancing. In the summary phase, the primary collects all active entries in state table into an FD structure which is constructed based on the Bloom filter which will be described later. During the update phase, a message is sent to the slaves by multicasting and a slave only saves the received data without any operation on its state table. The scheme shifts to the recovery phase after a failover, and the new primary reconstructs the state table in a packet-driven fashion by querying the stored FD structure (i.e., a backup SBF) to see whether an incoming TCP packet might be active classified by the old primary. If it seems true, the packet passes the recovery process. Otherwise, the new primary drops the packet.

II. OVERHEADS OF STATE REPLICAION

The dilemma of current SRP implementations should be highlighted; the state replication is necessary but high costs incurred to maintain the state consistency may impede its use as the cluster size increases or in a heavy-traffic environment. We found that both ct_sync and pf_sync send at least 3 messages for a well-behaved connection from SYN to FIN (i.e., an insertion, changes and a deletion). Because every state change in the primary activates an update message, the effort spent on updating a slave state table is proportional to the total number of received update messages. Consider a stateful firewall cluster operates at an Internet edge whose steady connection rate is 20,000 connections per second (cps). The connection rate here includes the connection setup and teardown rates. Assume the information used to state replication contains a flowID and the operation type whose size is about 100 bits, the size of output queue is large enough to hold all waiting messages and the update interval is 20 seconds. Then, an update transfer introduces a considerable 20,000×3×20=1,200,000 messages and this would also take 120Mb of memory space and network bandwidth to buffer and transfer these messages. Second, more overhead reduction implies more system capacity of a primary to process the regular pass-through traffic. Moreover, a state insertion in the primary also implies that a slave performs the same operation into its state table simultaneously. However, the state table of slave node is only of use when running as a new primary and attempting to recover the active connections. Obviously, a replication protocol to ensure state consistency by merging all modifications to a backup state table introduces a considerable overhead into a cluster.

To resolve the above problems, a new scheme for state replication is needed. Our goal is to lower as much update cost as possible in order to maximize the primary capacity and HA scalability. Thus, a new scheme called flow digest (FD) for state replication is proposed as follows.

III. FLOW DIGEST

A. Overview

In contrast to existing, FD improves the replication procedures by two factors: (1) the new primary only references the state table when it takes over the traffic processing and (2) a new data structure is designated to save bandwidth requirement. The FD scheme can be divided into three phases: summary, update and recovery. They are described briefly as follows.

In the summary phase, the primary collects all active entries in state table into an FD structure which is constructed based on the Bloom filter which will be described later. During the update phase, a message is sent to the slaves by multicasting and a slave only saves the received data without any operation on its state table. The scheme shifts to the recovery phase after a failover, and the new primary reconstructs the state table in a packet-driven fashion by querying the stored FD structure (i.e., a backup SBF) to see whether an incoming TCP packet might be active classified by the old primary. If it seems true, the packet passes the recovery process. Otherwise, the new primary drops the packet.

B. Bloom Filter Background

By using a bit vector $V$ of length $m$ and $k$ independent hash functions with range $[1,m]$, a standard Bloom filter (SBF) [10] yields an extremely compact and one-way data structure that supports the membership queries to a set $A = \{a_1, a_2, a_3, \ldots, a_n\}$ of $n$ elements in constant time. The Bloom filter causes the space requirement to fall significantly below the information theoretic lower bounds for error-free data structures and can reduce the space by at least one order of magnitude. It achieves this efficiency at the cost of a small false positive
rate, but has no false negatives. The term false positive describes the item not in the set is classified as being in the set in a query. The term false negative describes the item in the set is classified as not being in the set in a query. There is a tradeoff between the size $m$, the number $k$, and the possibility of false positive $f$ as the following Equation 1 and it will give a minimum value when $k = \ln 2 \times m/n$. Fig. 2 depicts the theoretical error rate with 4 or 8 hash functions and maximum active connections respectively versus different bit-vector sizes. For example, for $n = 1M$ and $k = 4$, if we choose $m = 1M \times 10 = 10M$ (bits), then the $f$ will be equal to 1.2%.

$$ f = \left(1 - e^{-\frac{kn}{m}}\right)^k \tag{1} $$

The SBF and its variants are widely used in practice when the storage is at a premium (e.g., the memory space is too valuable to store the large volume of data) or an occasional false positive is tolerable (e.g., [12]).

In the update phase, two message formats are used according to update sizes. In a high connection-rate environment, the primary sends the SBF to slaves directly to achieve lower update overhead. One can thus view an update as the snapshot of a state table propagating outward from the primary. As described before, except for shifting to the recovery phase, the state table in a slave will not be accessed, clearly quite different from current SRPs, and this prevents the slave from merging every incoming message its local state table in real-time. Every update by sending entire SBF is self-contained, so that the slave just replaces the stored data with a received SBF directly.

Sending an SBF (e.g., 4,096K bits) to slaves is clearly not economical at all in a slow traffic environment (say, below 1,000 cps), because the update information contained in an SBF may be only slight different from the one before it. This makes the incurred overhead exceed the benefits of the proposed method. An alternative is to use a difference mechanism which forms an update message (called difference message) issuing changes. A difference message is composed of a list of 32-bit entries and every entry uses the most significant bit for specifying whether the bit should be set to 0 or 1 and the rest bits for specifying the SBF index to be modified. The choice of which message format to use will depend on what the size of an update will be. Obviously, if the difference between two updates is small, it is more saving to use difference messages rather than entire SBF.

After a failover, a new primary comes up and enters to the recovery phase. It bases on the backup SBF from the old primary to process the incoming traffic and reconstruct its state table in a packet-driven fashion. A SYN packet will not be filtered by the backup SBF. When there is a non-SYN TCP packet arriving and it can not be found in a state table lookup, the new primary performs a membership testing on the backup SBF and the bits corresponding to the result are checked. If all bits are positive, then the new primary accepts this connection.
Two possible errors in FD are defined:

- False hit: A connection is not active for the old primary, but the backup SBF answers a positive for the query.
- False miss: A connection is active for the old primary, but the backup SBF indicates it is not.

Two possibilities for cause of a false miss: the false negative from the backup SBF and the state inconsistency between the primary and slave. The problem of state inconsistency will be discussed later. In FD, using a counting Bloom filter as the representation of active connections incurs small false positive and false negative rates. The false miss affects the performance of pass-through TCP traffic, because the packets of a misclassified connection will be dropped continuously in general, including all re-transmissions. In order to reduce the probability of yielding an overflow event (the cause of a false negative), the counters in our array need to be large enough to avoid the counter overflows. On the other hand, the counter size needs to be made as small as possible to save main memory. According to the analysis of [11],[12], it reveals that 4 bits per counter should be sufficient for most applications, so do the counters of a CBF.

Occasionally, a Bloom filter may return false positives. Note that the false hits do not affect the recovery of active connections. But, this means that a packet without the previous 3-way handshake procedure passes through the filtering of new primary because the SBF returns a positive answer in the recovery phase. A network attack may use this as ACK floods to exhaust the system and network resources of victims. The packet rate of attack was reported as high as 200,000 pkts/s [18],[19]. In the following, we briefly discuss how to minimize the syndrome of false positives.

First, since the packet reaches the particular endpoint through a false positive, depending on the implementation of OS, the endpoint either replies back a TCP RST packet or an ICMP unreachable packet to sender and this rogue ACK packet will be discovered eventually when the new primary receives this network error packet. Second, a DDoS prevention module is popularly equipped in a modern firewall or IPS. When the ACK flood is detected, for optimizing the utilization of state table, the new primary can enable the aggressive aging [15]. Third, it is noteworthy that false positives only exist after failing over and the duration of this potential risk is short. Finally, the failover timing and the parameters of an FD implementation should not be predictable and open to network attackers. Therefore, we believe that it is difficult for attackers to use the feature of Bloom filter to bypass the security filtering, especially when the system uses DDoS detection as a front-end to prevent the state table explosion [16].

A third kind of error introduced into HA comes from the overhead of failover process, including the failure detection and primary election. Instead of an efficient state replication, a fast network-level failover mechanism is required to minimize failover overhead.

The number of hash functions influences two competing forces: the probability of collision and the capability of the discrimination between flow IDs. Besides, in FD, a larger number of hashes may increase update size in a difference update. We compare three hash functions. First, MD5 is an open and well established message digest algorithm and is chosen for its well distributed and fix-sized output values. Four hash values are built by calculating the MD5 checksum [14] of a flow ID, which yields a 128-bit data, then dividing it into four 32-bit words. The indexes into the SBF and CBF come from applying the modulus to resulting 4-byte values by size m finally. Second, a modification of MD5 in [13] (called MD- by ignoring the G, H, and I functions) is used to improve the throughput and has a close result with MD5. Third, we use a “fast” hash family based on the shifting, AND, and XOR operations on flow IDs. Although this solution is less effective than MD5 and MD- in avoiding collisions, it is characterized by much lower computational overheads and can be implemented by a limited and simple instruction set (e.g., RISC-based network processor units).

D. Update Criteria

In order for the recovery phase to maximize the reconstruction performance, the backup FD at each slave must be kept most up-to-date with the primary state table of the primary. Ideally, the primary should continue to propagate the last information on itself. However, this increases the update overhead, because frequent propagations of update messages incur a non-trivial cost on system performance and should be avoided. The key to the scalability of an HA cluster is to ease the load on the system and network. Thus, a backup SBF without updated in real-time helps the scalability; rather, the update can be triggered upon simply a periodic basis, or a threshold basis. These two methods are transparent in operation. The update can occur upon a regular time interval, or when a certain percentage of the variation on existing connections compared to the previous update is not reflected to a slave. That is, FD uses an occasional message burst for providing a table snapshot to replace the continuous small messages for updating state changes.

E. State Inconsistency on Active Connections

Because a failover is likely to be prior to an update event and results in a state inconsistency, a periodic or threshold-based update method poses a potential risk to a new primary of dropping the packets of active connections in the recovery phase. Two approaches are used in the recovery phase to solve the problem: TCP cold start [17] and the intentional block mechanism.

TCP cold start lies on the assumption that the new primary lies between a trusted network (e.g., internal network) and an untrusted network (e.g., external network). If a non-SYN packet which fails on lookups both in the state table and the backup SBF comes from the trusted network, it will pass the recovery filtering and the state entry will be instantiated into the state table. However, if the packet is from the untrusted network, the new primary forwards the packet to the destination by stripping off its payload and decreasing the sequence number in the header. In this way, if the packet is
Indeed from a reliable connection, then the endpoint in the trusted network will respond this "keep-alive" packet with an ACK. Then, because the ACK comes from the trusted network, the new primary instantiates the corresponding state entry and continues as usual.

Using the intentional block mechanism, for the same packet above, the new primary inserts the corresponding flowID into the backup SBF regardless of packet direction and simply drops the packet. Again, if the connection is active, the source endpoint will initiate a retransmission. This time, the new primary will get a positive answer from the backup SBF for the retransmitted packet and forward the packet to its destination.

In summary, by taking advantage of the state held at the receiver and the sender, in the recovery phase the new primary can continue the active connections which are lost in state replication. While supporting these methods introduce additional bandwidth consumption on the pass-through path, the duration of recovery phase is short and, more importantly, these schemes relieve the load in propagating the update messages immediately for state consistency and allow a larger update interval or threshold to reduce the network overhead.

![Figure 4: The average bandwidth consumptions under different connection rates of SRP and FD (m=4Mb, m/n=4). The update interval is 5 seconds. Note that the update of FD is by sending entire SBF.](image)

**Figure 4.** The average bandwidth consumptions under different connection rates of SRP and FD (m=4Mb, m/n=4). The update interval is 5 seconds. Note that the update of FD is by sending entire SBF.

![Figure 5: The average bandwidth consumptions in log scale under different connection rates of SRP and FD by different update intervals. Note that the update of FD is done by sending entire SBF and the y-axis is in log scale.](image)

**Figure 5.** The average bandwidth consumptions in log scale under different connection rates of SRP and FD by different update intervals. Note that the update of FD is done by sending entire SBF and the y-axis is in log scale.

### IV. Evaluations

This section presents the results of simulation (written in C++) to evaluate network costs of FD and SRPs from low to high connection rates. We consider a stateful cluster composed of a primary and a slave and this HA cluster processes the pass-through traffic from internal and external networks. The node-to-node link is 100Mbps which is chosen to model Fast Ethernet and all updates are delivered by unicast. In this topology, the range of steady connection setup and teardown rates and active connection duration are 500 to 90,000 cps and 5 to 30 seconds, respectively. The setup delay (the time elapsed between the first SYN to the first ACK) is set as 2 seconds [16]. We study 256K, 512K, and 1M active connections and simulation time is 3,600 seconds.

We experienced with three FD configurations: 1Mb, 2Mb, and 4Mb SBF sizes for supporting 256K, 512K, and 1M connections respectively. A CBF counter takes up 4 bits. FD has 4 hash functions by calculating 128-bit MD5 outputs from an open source library. Based on the connection rate and active connection number, one can convert the thresholds to time intervals, hence, for FD, we only use a periodic update method and the update interval is from 1 to 30 seconds. Two update message formats of FD are both compared with current SRPs in the simulation. Following the ct_sync and pf_sync, the simulator for current SRPs sends messages for state insertions, changes, and deletions, and message size is 13 bytes which contains one flowID and a flag to specify update type. We do not consider the overflow of a delayed output queue and all messages of current SRPs are sent to the slave in real-time.

We first show the estimated update cost caused for different connection rates in Fig. 4 and 5 which indicate the bandwidth consumptions of current SRPs and FD by sending entire SBF. Fig. 4 shows that the FD structure to support 1M connections with 5-second update interval eliminates 46%, 73%, 86%, and 97% of bandwidth consumption compared to current SRPs at 5,000, 10,000, 20,000, and 90,000 cps, respectively. In Fig. 5, we observe that FD by sending entire SBF provides a snapshot of the current state table at a moment and the message size is deterministic for every update regardless of different connection rates. By contrast, the total number of update messages under a connection rate does not change, thus current SRPs have fixed costs on network depending on the connection rate of regular traffic. Therefore, FD requires a less bandwidth as the update frequency decreases. For example, if the update interval is set to 30 seconds, FD reduces the bandwidth consumption by 91% to 99% for 1M connections. Thus, the network overheads of the proposed scheme can be reduced significantly with a larger update interval. In addition, the network overhead can be improved due to a smaller maximum connection number \( n \) with the same \( m/n \). For example, in Fig. 5, it shows that FD reduces at least 86% of the bandwidth consumption with 5-second update interval both for supporting 256K and 512K connections. Second, we also simulate FD with difference update with low connection rates (not shown in the figure). When the primary works in a slow traffic environment or with a small update interval, e.g., 2 seconds, a difference mechanism is used because the update size is less than \( m \) bits.
Compared to current SRPs, the simulation results demonstrate that FD with difference messages eliminates 20% of the network bandwidth and reduces the number of update messages by 34% to 37% from 500 cps to 5,000 cps. The above results explore that the major benefit of the proposed scheme is to improve the bandwidth utilization, especially at high connection rates.

We study the throughputs of MD5, MD- and fast hash functions by getting 324M sets of hash outputs ($k=4$) without other operations. Our results show that the throughput of fast filter and MD- filter achieve about 5.87 times and 2.45 times higher throughput than MD5 filter. Our simulation tests also demonstrate that the hash function dominates the performance of the proposed scheme.

V. DISCUSSIONS

There is essentially no limit to how many nodes can participate in a cluster and the network and memory overheads introduced by maintaining state consistency determine the scalability of a replication protocol. The bandwidth needed by current SRPs scale linearly with the amount of regular traffic passing through the HA cluster and the number of update messages may grow up to many thousands per second (e.g., at 20,000 cps). By contrast, the number of bytes needed by an update is at most a constant value ($m$ bits) and the number of update messages can be determined by update interval and is smaller than current SRPs. As our simulation results show, the proposed method requires relatively much less network bandwidth by sending entire SBF and less update messages by sending difference messages and therefore is more scalable for state replication than existing methods both for low and high traffic loads.

Notice the parameters for FD: the number of hash functions $k$ and type (e.g., MD5 or fast hash), a bit vector of length $m$, additional bits for each CBF cell, update approach, and $n$ simultaneous maximum connections. These parameters are negotiated at HA initialization time or when a new node is trying to connect to a functioning cluster. Recall that the feature of Bloom filters is that they provide a tradeoff between the storage requirement and accuracy. Thus, if one wants to run with less bandwidth consumptions for state consistency, this can be achieved by reducing the size $m$ with possibly slightly increasing of entries into the state table because of more false positives in the recovery phase. The other approach to reduce network overhead is to choose an appropriate larger update interval according to the traffic conditions. Furthermore, by the above settings, FD occupies a total space of $(m + mb + m)$ bits at most in memory of a primary, i.e., an FD data structure and difference messages of size $m$ at most, $m$ bits at most to the network for individual update by sending an SBF or a bulk of difference messages, and $m$ bits in a slave to store a backup SBF.

VI. CONCLUSIONS

For a stateful HA scenario, the current solutions use update messages for state replication which may use a substantial amount of network bandwidth and this extra overhead could also reduce the capacity of a cluster to process the regular pass-through traffic. Moreover, the computation requirements associated to maintain the state consistency between the cluster nodes result mainly from the processing of update messages and merging the state changes into the local state table. In this paper, a scheme called “flow digest” has been proposed to improve existing state replication protocols by reducing the update overhead for both low and high connection rates.

The main advantage of the proposed method is to reduce the network overheads of state replication. The simulation results show that the bandwidth consumption of the flow digest is much less than that of current implementations. The proposed scheme by difference messages eliminates 34% to 37% of the network bandwidth and reduces the number of update messages to the slave by 20%. More importantly, at high connection rates, the bandwidth consumption can be reduced typically by at least 86% compared to current solutions.

REFERENCES