A Centralized State Repository Approach to Highly Scalable and High-Availability Parallel Firewall

Kasom Koht-arsa
Kasetsart University, Bangkok, Thailand
Email: Kasom.K@ku.ac.th

Surasak Sanguanpong
Kasetsart University, Bangkok, Thailand
Email: Surasak.S@ku.ac.th

Abstract—**Conventional high-availability stateful parallel firewall suffers from low scalability due to two overlapping requirements: workload distribution and redundancy. To achieve high throughput, load-distribution with complex algorithm is conventionally employed, consuming a lot of resources and making the system susceptible to state-related attacks such as SYN-flooding. On the other hand, making the system redundant usually implies N-to-N cross-replication of connection-state data among firewall nodes. These make the scaling effort very difficult at best.**

This paper presents the novel design and implementation of a highly scalable, high-availability, stateful parallel firewall with centralized state repository intending for high-speed connection environment. The system consists of fault sensor unit(s), fully redundant load manager units, fully redundant central state repository unit(s), and an array of Linux-based machines acting as firewall nodes under the data parallel scheme. Adding more units into the system can scale every component up. Consistent Disjoint-subset Hashing and Stateless Load balancing algorithms, chosen for their superior computing overhead, provide high performance, flexibility and scalability. Centralized State Repository further enhances reliability and scalability.

Actual deployment statistics confirm that the combination of centralized state repository and on-demand state restoration largely reduces the number of state synchronization transactions when the number of firewall nodes fluctuates. Therefore, the high-scalability and load balancing are gained with minimal state replications.

Index Terms: **firewall, stateful firewall, parallel firewall, high availability, fault-tolerant, fully redundant, scalable, state replication**

I. Introduction

Firewalls are widely deployed as security mechanism to provide access control at the border of networks. While network connection is rapidly evolving toward high-bandwidth capability in scale of terabit per second, firewall performance is a major concern. A single firewall tends to be a system bottleneck due to packet inspection process and complex rule sets matching.

Typically, a single firewall cannot accommodate a large volume of packets generated from high-speed connection. Parallelism is an approach to build high performance firewall that offers scalability and high availability to network services. A parallel firewall consists of array firewall nodes that operate independently and concurrently in parallel fashion. Load balancing is required to distribute traffic through firewalls, so that each firewall fits the whole or part of packet filtering policy.

Node array also offers enhanced redundancy and availability. When a node in the array fails, the remaining nodes can take over the jobs assigned to the failed node, under predefined automatic fail-over mechanism.

State synchronization is an underlying mechanism to provide redundancy. Parallel firewalls use automatic fail-over mechanism to replicate TCP connection states from a failed node to a redundant node. A node in stateful parallel firewall [7] generally relies on active-active mode where any change in connection states must be replicated to the rest of nodes [9][10][11][17][18]. This is to ensure that at least one of candidate nodes will take over the connection states of a failed node and continuously operate transparently and seamlessly to users. However, this synchronization often induces large amount of traffic because of unavoidable state replication among nodes. Thus, high performance and true scalability cannot actually be achieved in conventional firewall.

Results from analysis and measurements show that both load balancing gain and scalability can be accomplished with minimal state replications. The paper is structured as follows: Section II presents related works. Sections III and IV describe the proposed methodology and system architecture, respectively. Section V describes designs of load manager, a component for manage traffic flow. Section VI explains the implementation and evaluation. Section VII provides discussion and future work, and Section VIII concludes the paper.

II. Related Works

Ongoing research related to parallel firewall [9] focuses on transparent and seamless continuation of services despite individual node failure. Redundancy protocols such as CARP [8], VRRP [15] and HSRP [16] may be adopted as high availability mechanisms for parallel firewall with only a small number of nodes. These protocols are designed so that when only one node is active, the remaining nodes are left idle. Hence, it is ineffective for parallel firewall where all nodes in the arrays are required to be active. Furthermore, the "heartbeat" technique can only validate the availability of nodes, but not the availability of interconnections. This technique also lacks mechanisms to reconfigure the system when a node fails, as required in parallel firewall.

Prior works mostly focus on analysis of state-replication effect and explore methods to compromise between performance and high availability. Goddard et al. [14] analyzed the unavailability of firewall sandwich configuration and suggested a detection message method to identify node availability. A detection message is specially crafted packet containing information to identify a failed node and to reconfigure system corresponding to current available nodes. This method is more reliable than the heartbeat technique used in redundancy protocol technique [8][15][16].

A stateful firewall has to keep track of TCP states [7] for every connection. The ongoing established connection is called "active connection." To preserve the active connection in high availability stateful firewalls, the state replication is required. Different operating systems offer different state replication techniques, e.g., *pfsync* [13] in OpenBSD, and *ct_sync* [4] with *conntrackd* [10] in Linux. Neira [10] also showed performance measurement of state replication in primary and backup firewall, and in multi-primary firewall [11]. Feng[17][18] evaluated the TCP state replication methods and showed that relaxing or delaying the state replication requires a much lower overhead and only small percentage of the traffic is lost.

One of major concerns in existing methods [1][4][10][17][18] is that every node in the array needs to hold the whole connection state table within itself. Any changes in number of nodes in parallel firewall, through failures, expansion, or administrative actions, will affect the state connection and require state resynchronization. Typically, state synchronization between multiple parties generates high volume traffic. This circumstance leads to performance degradation and inhibits scalability of parallel firewall.

The firewall sandwich methods [1] employ a simple hashing scheme. Its major weakness is that adding or removing a node severely changes hash results. The consistent hashing algorithm [2] proposed to solve the similar problem on the URL of web caching seems to be a good candidate. However, the URL hashing on web cache takes smaller fraction of both time and storage compared to the other operations of web cache, e.g., fetching data through the network, disk storage. Directly applying consistent hashing is not generally suitable for parallel firewall; hence, minor modification of consistent hashing is adopted for balancing traffic. This modification will be described in Section VI.

Balancing workload is crucial to determine the performance of parallel firewall. In multi-primary hash-based method, it requires almost 50% more nodes to justify the load balancing overhead. Multi-primary sandwich relies on load balancer, and it must be stateful and scalable according to the number of nodes in the array.

The primary-backup or active-standby uses only one active firewall node, and the rest of nodes are idle. The utilization of this method is not optimal because some resources are unused. Using more firewall nodes does not improve the performance. On the other hand, the multi-primary multipath utilizes all firewall nodes. Nonetheless, due to the nondeterministic route of traffic, a complex and high overhead connection state exchange method must be employed.

In multi-primary hash-based, every firewall node receives identical packets and filters out all irrelevant packets (packets that should be processed by the other firewall nodes). To implement high availability, every firewall node must exchange their connection states with the other nodes. This increases overhead and impedes the ability to scale.

The multi-primary sandwich requires complex stateful load balancers, demanding high storage and processing resources. The redundant load balancers are needed because the load balancers themselves are stateful.

Obviously, performance for stateful parallel firewall considerably relies on the effective manipulation of global connection states on all firewall nodes. As previously mentioned, adding more firewall nodes does not reduce the number of connection states kept in each node. In contrast, more additional nodes induce the number of state exchanges between them. To reduce or remove computation overhead of nodes from load balancing function, we propose a simple stateless load management. Furthermore, fully direct state synchronization can be avoided with a centralized state repository. On-demand state restoration with centralized state repository substantially reduces number of state exchanges between nodes. However, this scheme still needs both state updates in a normal operation and state restoration in a fault situation. Compared to the previous works, our analysis shows that an on-demand state restoration with the centralized state repository result to smaller number of state updates and state restoration time.

This paper extends our previous work that originally appeared in [6] by 1) propose design and models of load manager, 2) describe details of modified consistent hashing, 3) add the recovery time analysis and 4) results from real workload are added.

III. Design Philosophy and Architecture

High availability firewall should be able to tolerate to any failure of system components, both internal and external. Redundancy is a key to high availability. Conventional redundancy architecture like active-standby lacks in resource utilization, while active-active in stateful firewall suffers from synchronization overhead.

Furthermore, the conventional firewall sandwich methods are not scalable due to connection state exchanges. This section presents design of parallel firewall system architecture that is high availability, efficient and scalable.

To achieve high availability, the proposed method is designed to support redundancy of every component in the system. The fault sensor and an active path detection algorithm ensure that the system can detect fault in every component and adapt the system to maximize utilization of available resource.

To minimize synchronization between components and gain scalability, we propose a centralized state repository to store the system's backup copy of connection states. This scheme replaces a conventional N-to-N firewall state exchange that causes a large amount of overhead. The state restoration of any failed node is on-demand that also reduces system workload.

A. System Architecture

The proposed parallel firewall is classified as data parallel model [9]. For this model, each node in the array is configured with identical rule sets for packet filtering, where packets (data) are distributed across nodes in the array.

At least three main issues must be considered to design parallel firewall. The first issue is how to perfectly balance the load of firewall array nodes with optimum resource consumption. The second is how to quickly detect and recover the fault with minimum packet loss, and the last is how to minimize the state exchanges between nodes. These issues can be solved by a combination of automatic fail-over detection with fault sensor, and dynamic reconfigurable stateless load manager with a centralized state repository approach.

Figure 1 shows the simple model of underlying system architecture. There are four main components in our proposed architecture: 1) Hash-based Load Manager, 2) Firewall nodes, 3) Fault Sensor, and 4) State Repository.

The load manager is responsible for packet flow management. It has one network interface connected to the external environment, and multiple network interfaces connected to each firewall nodes. Packets enter the load manager and are distributed among firewall nodes using hash function. Two load managers are placed in a sandwich-style. They act as packet distributor and packet aggregator depending on the direction of arriving packets.

Each firewall nodes will process different packets simultaneously based on hashing scheme from the load manager. In data parallel model, each node provides access control and auditing based on a security policy defined by identical rule sets.

The centralized state repository (CSR) is an active machine used to keep track of all connection state. The CSR approach is more efficient compared to the independently internal state bank in each node. Offloading unrelated connection states to the state repository results to lower overhead processing and less resource used in each firewall node.

The fault sensor has two special network connections (dash line) to the load managers on the upper and lower sides. As a result, the fault sensor can send and receive packets through the main external interface of parallel firewall.

Functional details of each component will be described in Section IV.

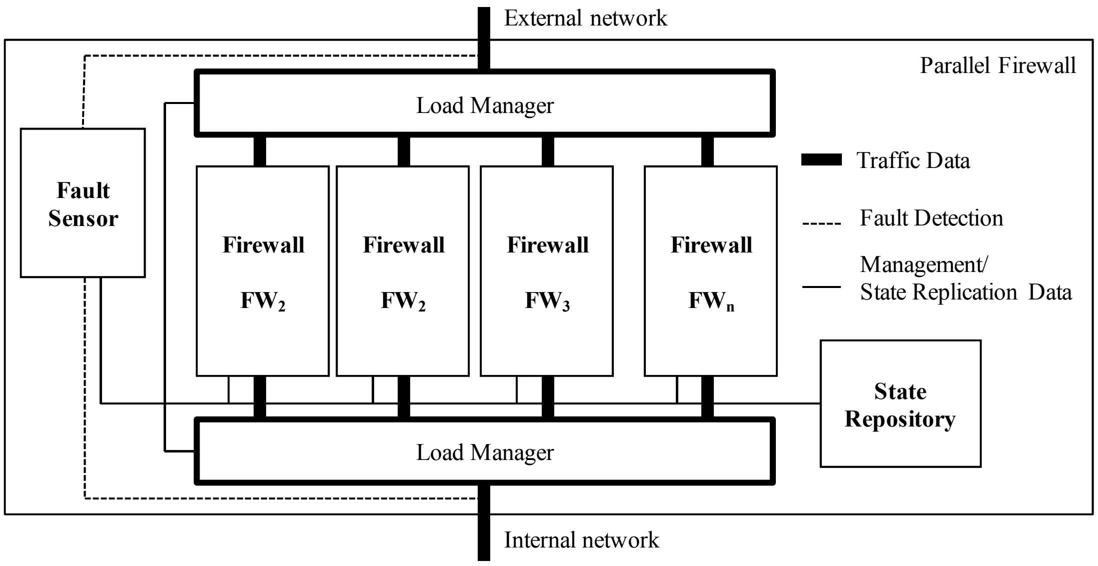


Figure 1. System architecture (minimum model).

B. Fully Redundant Model

Every component in the system architecture shown in Figure 1 except firewall nodes is potentially a single point of failure. To provide the fully redundant system, we append a copy of each component as shown in Figure 2.

The load managers on both upper and lower parts are stateless by nature. Each group of load managers will have identical hash parameters and connect to the same set of firewall nodes.

A firewall node has four interfaces, instead of two in order to forward packets configured as two bridge groups, with two network interfaces on each group. Each two-bridge group shares the same connection state table.

An example of a deployment for the full redundant model in real network environment is shown in Figure 3.

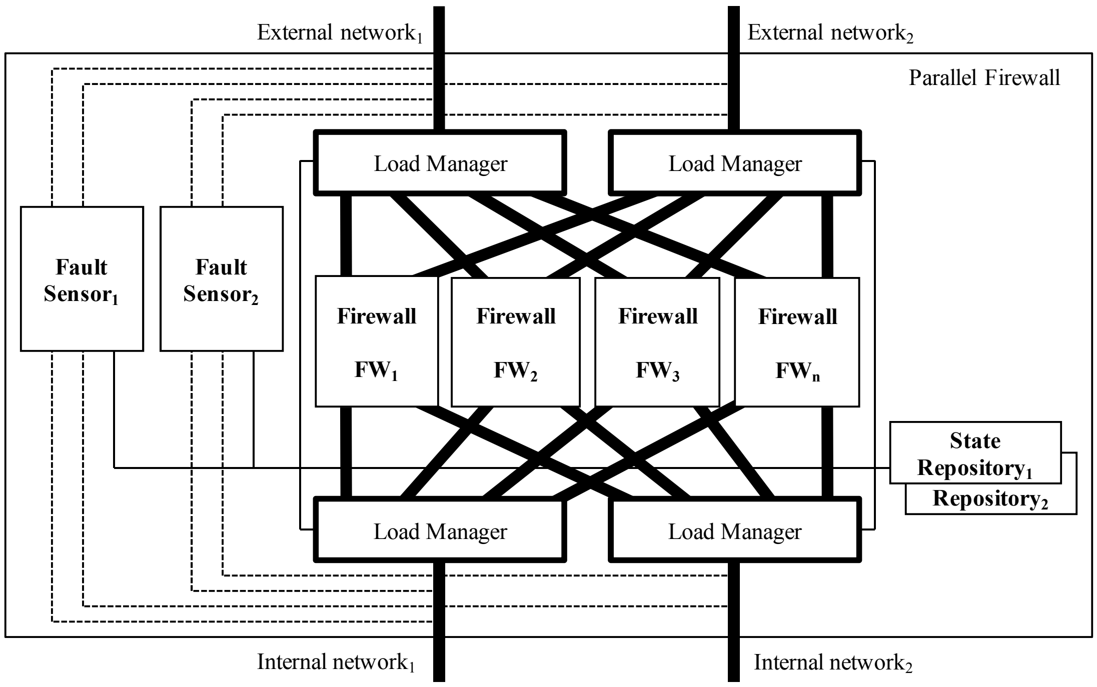


Figure 2. System architecture (fully-redundant model).

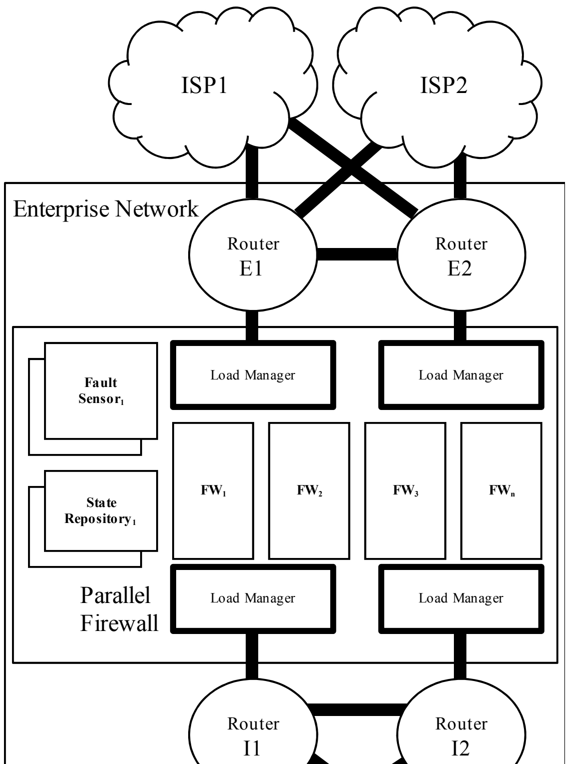


Figure 3. Typical system deployment.

IV. System Components

This section describes the four key components in the proposed models: 1) load manager, 2) firewall node, 3) fault sensor and 4) state repository.

A. Load Manager

A load manager is a crucial component used to distribute and aggregate packets to and from firewall nodes. Naturally, each firewall node will handle disjoint subset of traffic assigned by the load manager and maintain its own set of connection states. Conventional load balancer may not be suitable for applying as load manager for parallel firewall due to its stateful characteristics. Connection tracking for stateful load balancer requires high computing power resulting to limited throughput and high latency. From this scenario, we design the load manager as stateless component and incorporate hashing scheme to deterministically assign traffics to firewall nodes. An ASIC or FPGA chip can be used to implement wire-speed stateless load manager to gain high performance with low packet forwarding latency.

The hash function on the load manager plays an important role in balancing the workload and increases the efficiency of connection state exchange and restoration. As mention previously in Section II, output from typical hash function will be drastically changed when the number of nodes has been changed. This leads to tremendous state exchange between firewall nodes.

Applying modified consistent helps reduction of state exchange traffics.

Consistent hashing algorithm uses a regular hash function *h(x)* (such as MD5, SHA) to determine a point on the edge of a circle. It also maintains a list of random points that determine the final hash result. The list grows proportionally to the number of nodes and the hash results cannot be lookup directly. We replace the list of random points with a fixed size array of 2^m entries. Each entry stores the desired firewall node ID. The output of *h(x)* is trimmed using modulo function to match the array size. The trimmed result can then be used to lookup on the array. The modified algorithm is illustrated in Figure 4.

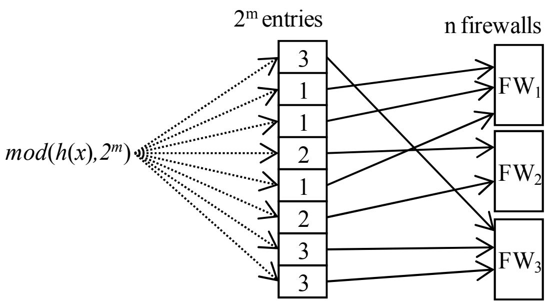


Figure 4. Modified consistent hashing.

The advantages of using lookup table instead of list of random points are:

1. The lookup has much lower latency than searching in a list.
2. If implemented in hardware, it does not require processing unit or any state machine.
3. It can be extend to support load balancing by adding a usage counter into every entry. This information is necessary for readjust the workload of firewall nodes.

To let every stateful firewall node operate independently without state synchronization, the load manager must ensure that traffic belonging to any two addresses always pass through the same firewall node. This constraint can be achieved by using source and destination addresses as the input of hash function. For example, if a packet is an outgoing packet, the source IP is used. Otherwise, the destination IP is used.

The load manager can be designed and implemented in many configurations. The models of how to implement the load manager are discussed in Section V.

B. Firewall Node

The CSR method requires at least two of firewall nodes running in bridging mode. Ideally, each node should have identical machine characteristics. Otherwise, the design of load balancing scheme must be complex to handle node inequity.

Each firewall node has two additional functions needed to interact with the CSR: 1) send connection state change events to the repository and 2) import connection state from the repository when the number of nodes is changed.

Let N_f be the number of firewall nodes and $T(f)$ be the throughput of node f. The CSR model can withstand a failure of N_f-1 firewall nodes with total achievable throughput T_{total} as shown in (1).

$$T_{total} = \sum_{f=1}^{N_f} T(f) \qquad (1)$$

C. Fault Sensor

The heartbeat-type methods employed in many redundant firewalls are not suitable to parallel firewall because they detect only the availability of the firewall management interface. The availability of this interface does not imply the availability of the data-forwarding engine of the firewall or the network connections. Instead, this work uses fault sensor to detect fault on any path of data flow. Moreover, this fault sensor injects/receives packet into/from internal/external network interfaces, using many test patterns to identify each path's availability.

For example, in fully redundant model with two firewall nodes, there will be four paths of data flow; each is bi-directional, as shown in Figure 5. The task of fault sensor is to apply the hash function to the load manager to create packets. These packets will traverse through each path to verify the path availability. In this case, eight packets are sent to verify four paths in two directions. When faults are detected, the fault sensors will adjust the hash function of load managers to adapt to the remaining available paths

.

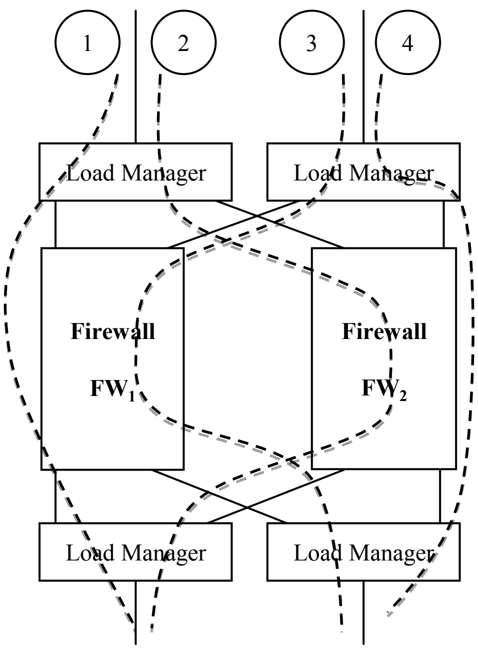


Figure 5. Example of four network paths in fully redundant model.

The fault sensor itself runs indefinitely. Its role is to detect the availability of every path and notify the other components when change is detected. The algorithm of the fault sensor is show in Algorithm 1.

Algorithm 1. Active path detection

```
active_paths ← $\varnothing$
while (true) do
   paths ← config file
   pkt_send ← $\varnothing$
   for all path $\in$ paths do
      add genpkt(path) to pkt_send
      add genpkt(reverse(path)) to pkt_send
   end for

   sendpkt(pkt_send)

   pkt_recv ← recvpkt(timeout_value)

   new_active_paths ← $\varnothing$
   for all packet $\in$ pkt_recv do
      if reverse_direction(packet) $\in$ pkt_recv then
         add path_of(packet) to new_active_path
         remove reverse_direction(packet) from pkt_recv
      end if
   end for

   if active_paths $\neq$ new_active_paths then
      update_hash()
      state-repository.update_hash()
      load-manager.reconfigure()
      active_paths ← new_active_paths
   end if

   wait for next interval
end while
```

D. Central State Repository

Synchronizing connection states is a limiting factor in scalability of conventional parallel firewall, because the conventional methods attempt to make copies of connection states on every firewall node. The larger number of nodes, the more states exchange workload. Instead of using state exchange, our method uses a state repository to store global connection state, as shown in Figure 6. These global connection states will be used only when the number of firewall nodes changes. Hence, our CSR method is more scalable than the conventional method because adding more firewall nodes does not increase workload on existing firewalls. The state repository has two main functions: 1) *update state*, called from every firewall node to update the global connection state, and 2) *update hash*, called from the fault sensor to change hashing parameters. The *update hash* function also notifies and sends connection state updates to the firewall nodes.

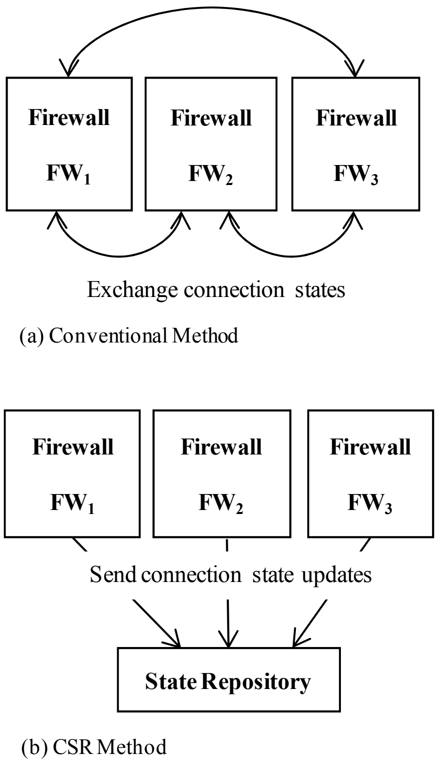


Figure 6. Comparison of state exchange between (a) conventional and (b) CSR method.

The difference between conventional methods and CSR approach is a methodology to manage connection states. In the conventional methods, every node has to keep track of all connection states in the system, so that it can recover when a node fault occurs. On the other hand, the CSR method keeps only relevant connection states at each node. A separate state repository is used to store the global connection states of the system.

In CSR method, when a node fails, the state repository will distribute portion of states previously belonging to the failed node to the remaining nodes. This mechanism is on-demand based. As shown in Figure 7, connection states from the failed FW$_2$ are transfer to FW$_1$ and FW$_3$.

When adding a new node to the system in conventional method, all current connection states must be imported to the new node. Either one node sends the whole copy or all of the former nodes partially send their copies to the new node. This export and transfer of states reduce the performance of the former nodes. In CSR, the state repository partially sends only states that belong to the new node. There is no overhead in the former nodes. This difference is illustrated in Figure 8.

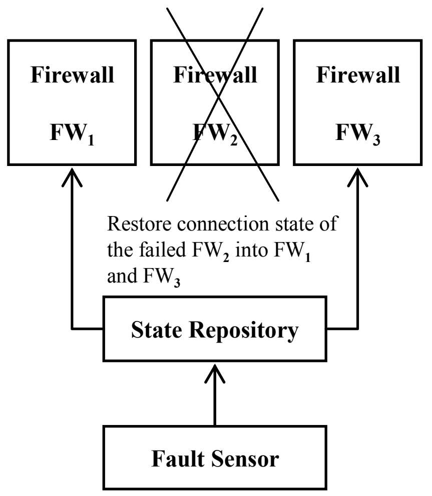


Figure 7. State restore from the repository. FW_2 is failed node.

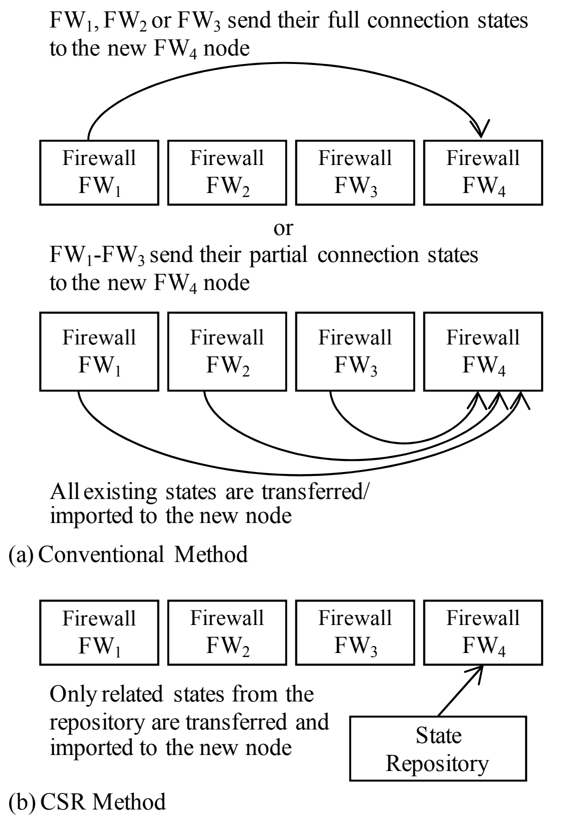
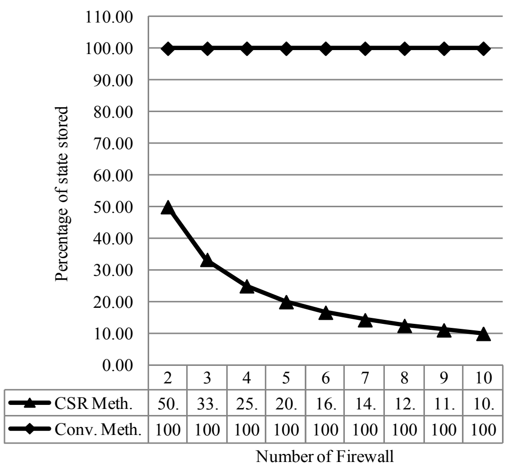

FW1, FW2 or FW3 send their full connection states to the new FW4 node
Firewall FW1
Firewall FW2
Firewall FW3
Firewall FW4
or
FW1-FW3 send their partial connection states to the new FW4 node
Firewall FW1
Firewall FW2
Firewall FW3
Firewall FW4
All existing states are transferred/ imported to the new node
(a) Conventional Method
Firewall FW1
Firewall FW2
Firewall FW3
Firewall FW4
Only related states from the repository are transferred and imported to the new node
State Repository
(b) CSR Method

Figure 8. Comparison of state restoration in (a) conventional method and (b) CSR method. FW_4 is new node.

Recall from Section IV 4 that N_s be is the number of all connection states in any given time. On average, each firewall node in the CSR method handles $1/N_f$ of the connections. Therefore, the average number of connection states on each node N_{sp} can be expressed as:

$$N_{sp} = \frac{N_s}{N_f} \tag{2}$$

N_{sp} becomes smaller as the number of nodes increases. Figure 9 illustrates the percentage of connection states reduction, in respect to the number of nodes from experimental. Note that in the conventional method, every firewall node holds all connection states, the number of connection states on each node of the conventional method is always equal to N_s and shown as 100% value.



	2	3	4	5	6	7	8	9	10
▲ CSR Meth.	50.	33.	25.	20.	16.	14.	12.	11.	10.
◆ Conv. Meth.	100	100	100	100	100	100	100	100	100

Figure 9. Comparison of percentage of states stored on each firewall. type.

The CSR method sends one copy of connection state to the state repository. From (2), the average number of states sent from each node is

$$N_{ssp} = \frac{N_s}{N_f} \tag{3}$$

Therefore, the accumulative number of states sent from every node is

$$N_{ssap} = N_{ssp} \cdot N_f = \frac{N_s}{N_f} \cdot N_f = N_s \tag{4}$$

On the other hand, the conventional method will have to send the connection states to all other nodes. Because each node in the conventional method handles traffic from N_s/N_f connections, the average number of states sent from each node from the conventional method is

$$N_{ssc} = \frac{N_s}{N_f} \cdot (N_f - 1) \tag{5}$$

And the accumulative number of states sent from every node is

$$N_{ssac} = N_{ssc} \cdot N_f = N_s \cdot (N_f - 1) \tag{6}$$

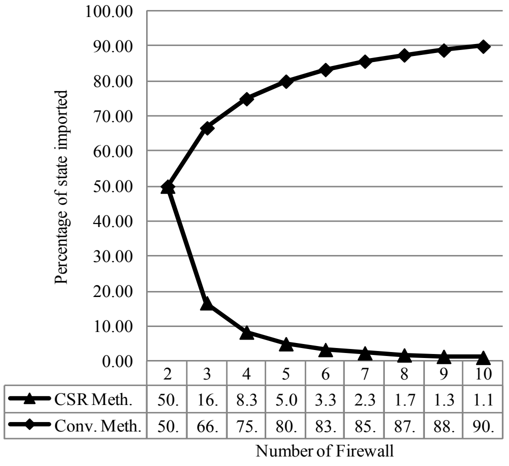
In CSR method, when one of N_f firewall nodes fails, remaining firewall nodes must import the state of the failed node. Let N_{sip} be a average number of state imported to each remaining node and can be expressed as:

$$N_{sip} = \frac{N_{sp}}{N_f - 1} = \frac{N_s}{N_f^2 - N_f} \tag{7}$$

The conventional method always imports states from the other nodes. The number of state import is equal to the number of connection states that are not passed through that firewall node itself. Therefore, the average number of state import on each firewall node in the conventional method is

$$N_{sic} = N_s \cdot \frac{N_f - 1}{N_f} \tag{8}$$

Figure 10 illustrates the percent of states imported on each type of firewall from (8) and (9).



	2	3	4	5	6	7	8	9	10
▲ CSR Meth.	50.	16.	8.3	5.0	3.3	2.3	1.7	1.3	1.1
◆ Conv. Meth.	50.	66.	75.	80.	83.	85.	87.	88.	90.

Figure 10. Comparison of percentage of state import on firewall nodes.

Because the on-demand state restoration occurs after the failure, there is some time gap between failure and full recovery. The total time required to fully recover (T_{rcvy}) from a fault depends on 1) the time to detect and find a fault (T_{df}), 2) the time to reconfigure load manager (T_{reconf}), and 3) the time to restore connection state (T_{rest}). T_{rest} depends on two parameters: time to transfer connection states from the state repository to a node (T_{xfer}) and time to commit the connection states into kernel's connection tracking table (T_{com}). Hence, the total recovery time can be expressed in (10) and (11).

$$T_{rcvy} = T_{df} + T_{reconf} + T_{rest} \tag{9}$$

$$T_{rest} = T_{xfer} + T_{com} \tag{10}$$

Conventional method has no T_{rest} because connection states are always up-to-date. In both the CSR and conventional methods, T_{df} and T_{reconf} are non-zero numbers; therefore, there will always be packet loss in both systems. The amount of packet loss will depend on T_{rcvy}.

T_{df} depends on polling interval, which is generally in scale of seconds. With a large number of firewall nodes, T_{rest} of the CSR method is very small (In an experiment, it is a fraction of seconds) because few states have to be restored. T_{reconf} and T_{rest} processing can be run simultaneously; thus, the value of T_{rest} is insignificant because it is far less than T_{reconf}. This makes recovery time of the CSR method similar to the conventional method.

V. LOAD MANAGER DESIGN CRITERIA

While the load manager is simple enough to be designed from the ground up, adapting commodity hardware may have more economic advantage. In this section, we propose two designs of the load manager. The first design adopts data access switches and the second one adopts commodity 802.1Q VLAN switch with additional simple FPGA hardware.

A. Data Access Switch Type

A Data access switch is typically used in IDS or traffic monitoring purposes. It provides several functionalities such as traffic aggregating from multiple links, traffic distributing to multiple ports, and traffic directing according to one-to-one and many-to-many port mappings. This section explains two schemes to apply data access switches for use as load managers.

Firstly, a data access switch with a full duplex feature (or receiving and sending function simultaneously within the same port) can be directly used for load balancing by setting aggregation and distribution rules according to the existing ports and the number of firewalls. This full duplex scheme requires two data access switches to cover the firewall array in a sandwich style as shown in Figure 11.

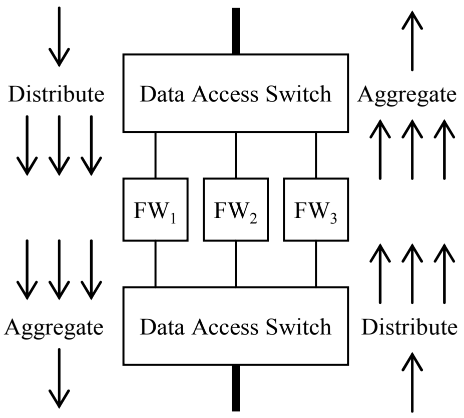
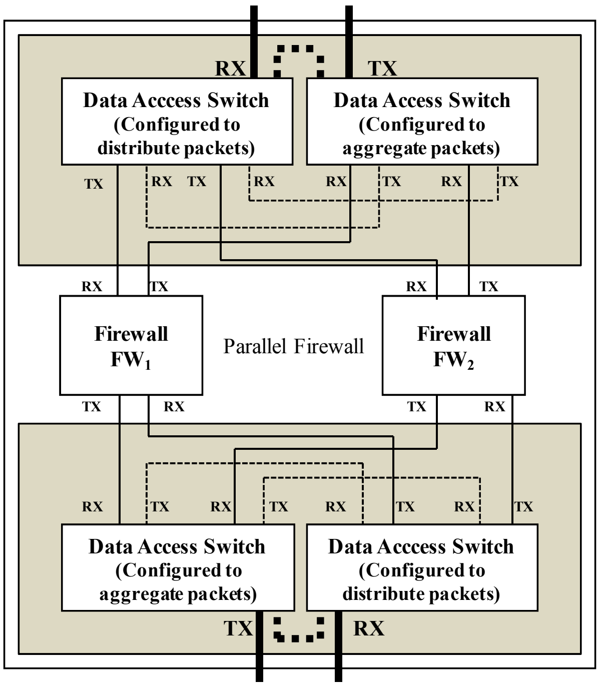


Figure 11. Using full duplex data access switch as load manager.

Secondly, a data access switch without a full duplex feature requires additional wiring connections. This scheme requires that all connections are optical fibers, with split TX and RX core [5]. Figure 12 illustrates the wiring connections with transmitting and receiving paths in details. The data access switches are configured to distribute and aggregate packets in pair. The dash line indicates that that link does not contribute to forwarding data, but may need to be connected if the data access switch cannot manually force the interface's status to active state.



RX
TX
Data Acccess Switch (Configured to distribute packets)
Data Access Switch (Configured to aggregate packets)
Firewall FW$_1$
Parallel Firewall
Firewall FW$_2$
Data Access Switch (Configured to aggregate packets)
Data Access Switch (Configured to distribute packets)

Figure 12 Connection diagram for data access switches that do not support full duplex operation.

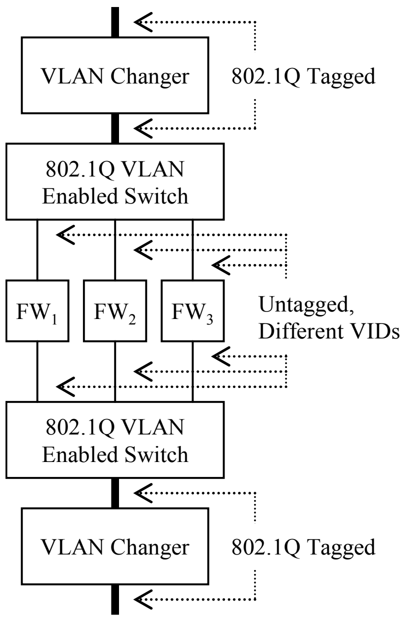


Figure 13. Connection diagram of VLAN Changer.

B. VLAN Switch Type

Virtual LAN (VLAN) is a mechanism that partition single physical switch into multiple logical switches. Commodity VLAN enabled switches are economical and almost all of them are capable of forwarding packets at wire-speed for each port. The concept behind making load manager from commodity VLAN switch is that the complexity and the cost of load manager is reduced by using the existing switch's packet buffer and forwarding fabric. An external component that control how the packets are forwarded is added to commodity switch to form a load manager. This method looks very similar to "router on a stick" configuration that a packet from one VLAN gets routed to another VLAN by a router, but it differs in how the destination VLAN is chosen.

To build the load manager, we propose a device called "VLAN Changer". The VLAN Changer is an FPGA with two Ethernet ports. Two VLAN Changers have to be placed in sandwich to the switches, as shown in Figure 13. To simplify the design of VLAN Changer, both ports of the VLAN Changer must be trunk ports. With this configuration, the incoming and outgoing packet will differ only in the VID and FCS (frame check sequence) field.

The VLAN Changer's control logic is a simple state machine that only determines the designated VID of any input packet and then changes the VID field of that packet accordingly. This part is easy to implement in FPGA. The complex parts such as crossbar switch or packet buffer are not needed because these parts are assigned to commodity switches.

VI. Implementation and Evaluation

We deploy the testbed on Kasetsart University's network (NontriNet). The NontriNet connects to two Internet Service Providers (ISPs); the first ISP has three of Gigabit Ethernet links while other has one Gigabit Ethernet link, as shown in Figure 14. The internal connections are 10 Gigabit Ethernet. To test on our deployed network without disturbing normal operation, we create the main parallel firewall similar to [5] and duplicate traffic flow feeding into the test firewall nodes. Such topology is shown in Figure 15.

The testbed has totally six firewall nodes. Four of them are used to forward traffics, while the other two are test nodes. All nodes have identical hardware specification: Dual-core Xeon® X5270, 4GB RAM, 140 GB SAS hard disk , and four gigabit Ethernet interfaces. The software is Linux kernel 2.6.29.1's with built-in Netfilter. The fault sensor and state repository have identical hardware, each which two dual-core Opteron^{TM} 2200 with 4 GB RAM and four gigabit Ethernet interfaces.

While the external links have total of 4 Gbps in and out bandwidth, the actual average network usage is 1.2 Gbps in and 511 Mbps out. The peak usage is 2.6 Gbps in and 1.4 Gbps out.

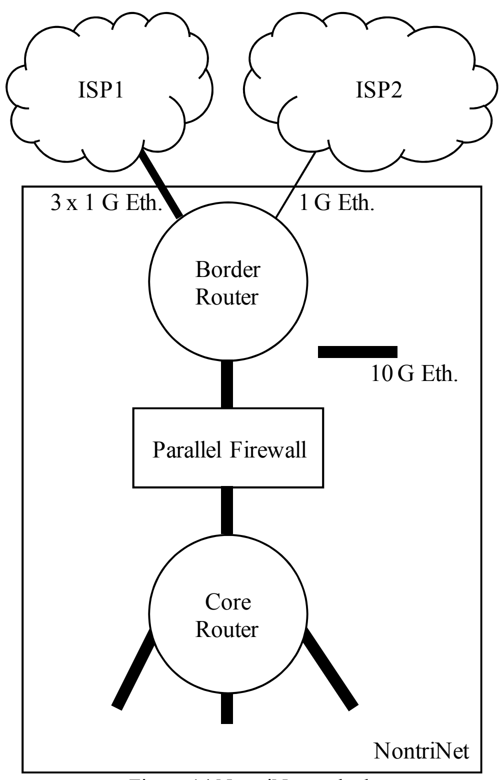


Figure 14 NontriNet testbed.

We adapt two data access switches model GigaVUE-MPs from Gigamon [3] as load managers. Because the GigaVUE-MPs do not directly support custom hash function, we create rules to simulate the effect of custom hash functions. The load managers are configured to duplicate traffic of FW_3 and/or FW_4 into FW_x and/or FW_y, depending on the test criteria. While the input packets are duplicated, the output from FW_x and FW_y are discarded.

In the first experiment, we measure the overhead of kernel state change export mechanism. The same set of traffic from FW_4 is fed to FW_x and FW_y. FW_x has *conntrackd* run in NOTRACK mode while FW_y does not. Using *cyclesoak* (a utility for measure CPU utilization) the measured load average on FW_x is approximately 12% larger than FW_y.

In the second experiment, we measure the overhead of exporting connection states to the other nodes, compared to full connection state exchange. The measured load average of full connection state exchange is about 14% larger than of export state only. The *conntrackd* has an external cache to store the connection state in memory without committing directly into kernel. If the external cache is enabled, the overhead is reduced to 5%. Nonetheless, it takes time to commit these connections later when needed.

In the third experiment, we measure the time to commit connection state into kernel space (T_{com}). It takes 1.2 second to commit 128,248 state entries into connection tracking table. The commit time grows linearly with the number of state entries. The time used to burst transfer 150,000 states from the state repository to a firewall node (T_{xfer}) is less than 5 milliseconds.

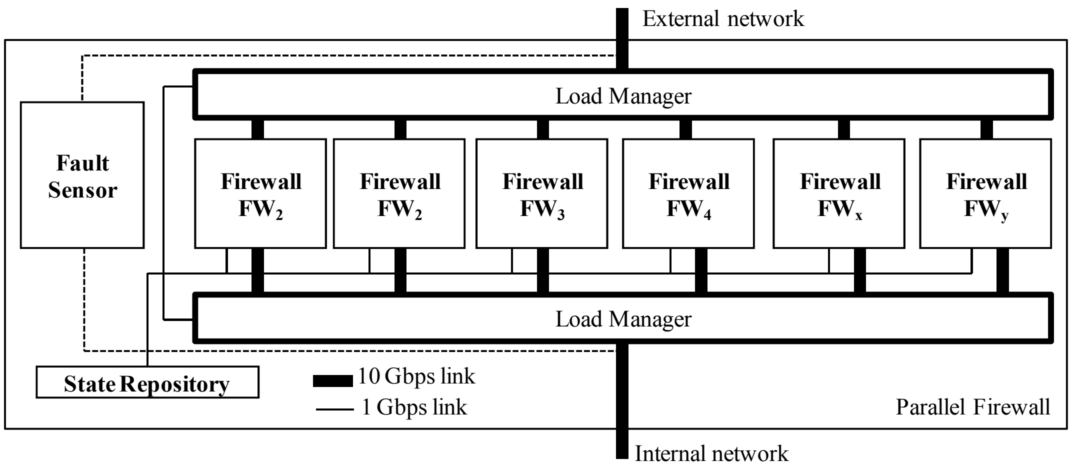


Figure 15. Configuration under test.

The real workload on each firewall is shown in Figure 16. The data point is five-minute average each, taken on 2012/09/25. Figure 17 shows the number of states on each firewall node. From those figures, the hash function distributes balanced workload to the firewall nodes. The bandwidth and number of states of each firewall node are different. Due to the nature of network usage, some clients may use more or less bandwidth than the others, and some clients may burst transfer traffic from time to time. The relatively light usage during hours 3:00AM to 8:30AM makes the difference in percentage standout. The imbalance of bandwidth and number of states during light load period is not important because firewall nodes also have plenty of processing resources left.

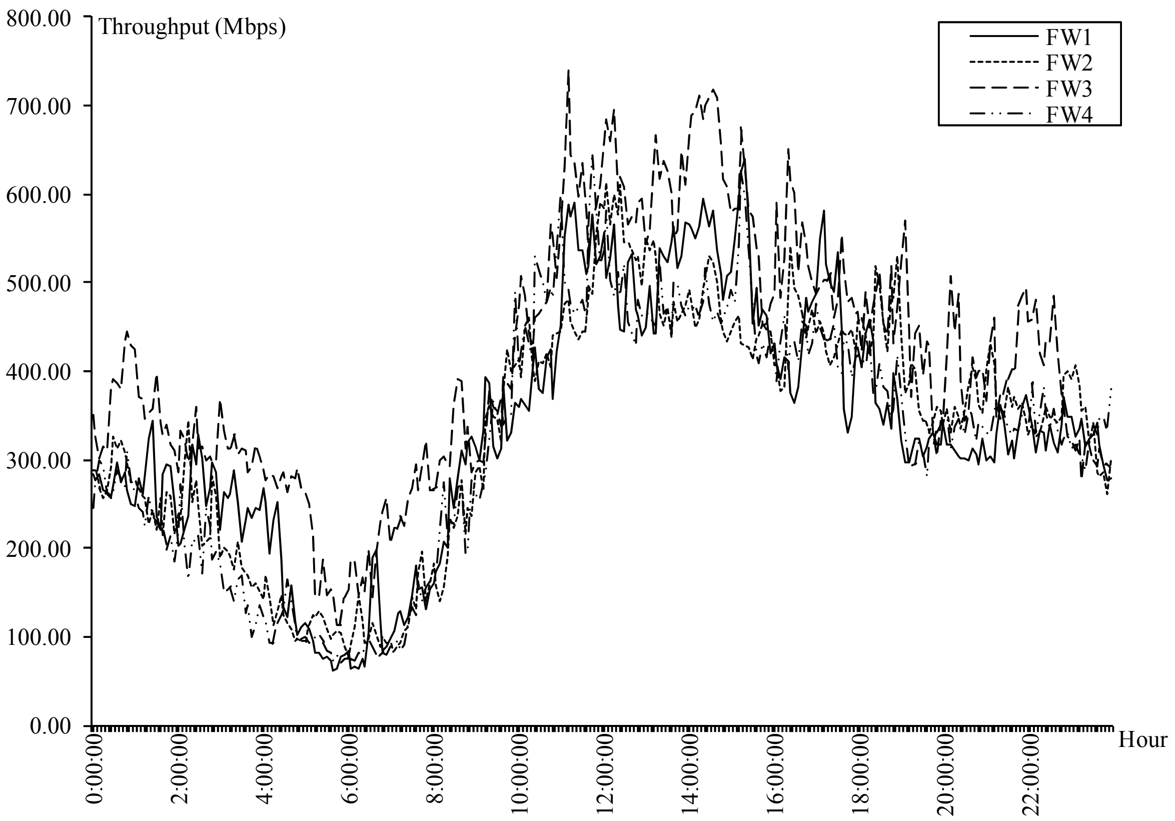


Figure 16. Throughput of FW$_1$ -FW$_4$.

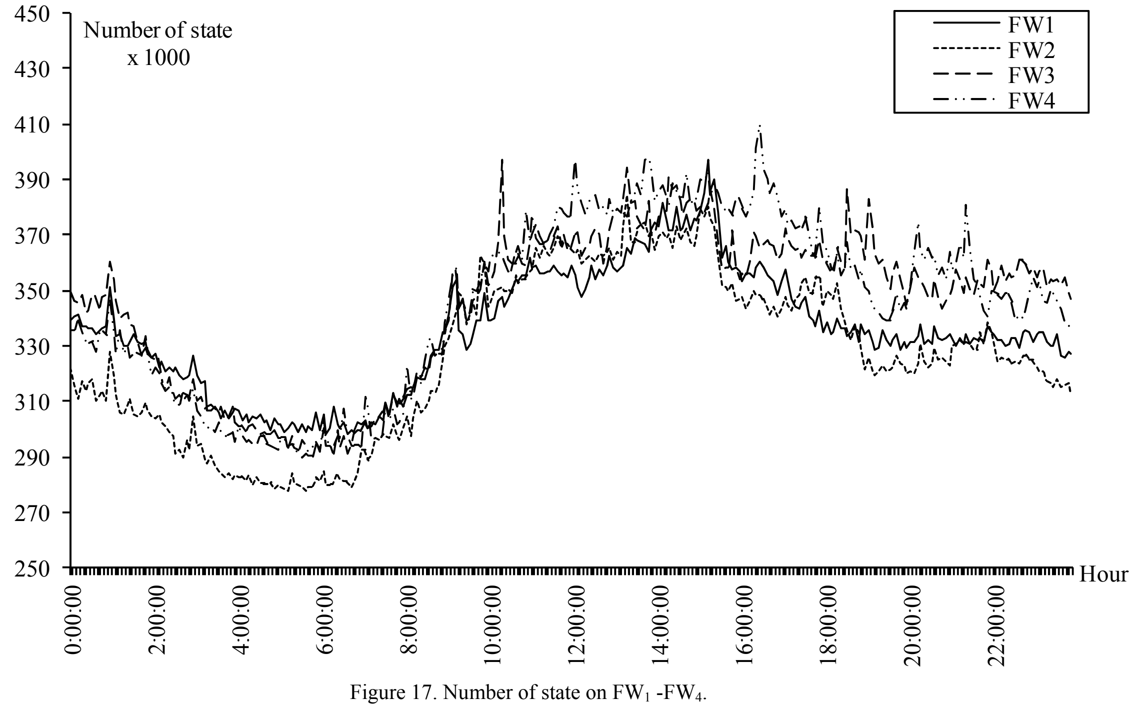


Figure 17. Number of state on FW$_1$ -FW$_4$.

VII. Discussions

The proposed CSR method is more efficient and scalable than the multi-primary hash-based. In multi-primary hash-based, every node receives all packets before most of them are dropped later after the hash filter. Adding more firewall nodes does not increase the receiving and hashing input packets. This results to diminishing return.

While the multi-primary sandwich does not have the same receive-then-hash overhead as the multi-primary hash-based, the load balancer itself is the bottleneck. Because the load balancer is stateful, it is not trivial to implement a wire-speed load balancer. Redundant load balancer is also difficult to design due to the demand of high-speed state exchange.

The CSR method replaces the stateful load balancer with a lightweight and much simpler stateless load manager. The redundant stateless load manager does not need state exchange.

The CSR method is more scalable than the conventional method. From (2), adding more firewall nodes does not cause more states stored on each firewall node. This results to lower resource usage in both space and processing power. Furthermore, comparing between (3)(4) and (5)(6), the CSR method send less number of states. In addition, (7) and (8) show that: 1) the state restoration occurs after the failure and causes no overhead during normal operation, and 2) the number of imported states is reduced because of the larger number of nodes.

In normal operation, the CSR method has 14% lower overhead than the conventional method. From (7) and (8), the more difference in overhead can be predicted when increasing the number of nodes.

The CSR method is more immune to state-related attack, such as SYN-flood. In conventional method, there are two major weaknesses from such attack, i.e., 1) the SYN-flood states will degrade the whole system performance due to a large number of state exchange between nodes, and 2) connection-state table in the load balancer will be rapidly exhausted, and cause a denial of service of the whole system. The design of CSR solves the above two weaknesses where 1) the flooded states will only be sent to the state repository without effect on other firewall nodes 2) the load manager simply ignores these states because of its stateless characteristic.

Future research includes improve the hash algorithm to support automatic workload balancing, design and optimization of VLAN Changer.

VIII. Conclusions

This paper describes stateful parallel firewall models that have high availability, high throughput, low latency, high efficiency and high scalability. The high availability is achieved by the ability to add redundant components to every part in the system. The added redundant components are all active and contribute to share the workload.

The proposed CSR method, which replaces the N-to-N state replication, reduces the system's overhead on normal operation and offers more system scalability. Moreover, the CSR method reduces the overhead and scalability problem of state replication by using on-demand state restoration and centralized state repository. The on-demand state restoration off-loads the overhead of state synchronization in normal operation. The centralized state repository makes the replication straightforward and has lower overhead than directly exchange connection states between firewall nodes.

Acknowledgment

This work was partially funded by a research grant from the Computer Engineering Graduate Program, Department of Computer Engineering, Faculty of Engineering, Kasetsart University.

The authors wish to thank Dr. Supaporn Erjongmanee and Asst. Prof. Dr. Pirawat Watanapongse for their valuable comments and suggestions.

References

[1] C. Benecke, A Parallel Packet Screen for High Speed Networks. Proceeding of the 15th Annual Computer Security Applications Conference, 1999. doi:10.1109/CSAC.1999.816014.

[2] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving HotSpots on the World Wide Web. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC'97), 1997*. doi:10.1145 /258533.258660.

[3] Gigamon, http://www.gigamon.com. Retrieved 2012/11/12.

[4] H. Welte, ct_sync: state replication of ip_conntrack. *Linux Symposium, 2004.* http://www.linuxsymposium.org/archives/OLS/Reprints-2004/Reprint-Welte-OLS2004.pdf. Retrieved 2012/09/30.

[5] K. Koht-arsa and S. Sanguanpong, A Practical Approach for Building a Parallel Firewall for Ten Gigabit Ethernet Backbone, *the 42nd Annual IEEE International Carnahan Conference on Security Technology, 2008*. doi:10.1109/CSST.2008.4751324.

[6] K. Koht-arsa and S. Sanguanpong, High Availability and Scalable Parallel Stateful Firewall Design, *International Conference on Internet Studies, 2012*.

[7] M. G. Gouda and A. X. Liu, A Model of Stateful Firewalls and Its Properties, Proceedings of the 2005 International Conference on Dependable Systems and Networks, 2005. doi:10.1109/DSN.2005.9.

[8] OpenBSD, The Common Address Redundancy Protocol (CARP), http://www.openbsd.org/faq/faq6.html#CARP. Retrieved 2012/09/30.

[9] P. Neira, R. M. Gasca, and L. Lefèvre, Demystifying Cluster-Based Fault-Tolerant Firewalls. *IEEE Internet Computing*, 13(6):31-38, December 2009. doi:10.1109/MIC.2009.128.

[10] P. Neira, R. M. Gasca, and L. Lefèvre, FT-FW: Efficient Connection Failover in Cluster-based Stateful Firewalls. Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008). Washington, DC, USA, 2008. doi:10.1109/PDP.2008.87.

[11] P. Neira, R. M. Gasca, and L. Lefèvre, Multiprimary Support for the Availability of Cluster-Based Stateful Firewalls Using FT-FW.*ESORICS 2008.* doi:10.1007/978-3-540-88313-5_1.

[12] P. Neira, L. Lefèvre, and R. M. Gasea, High Availability Support for the Design of Stateful Networking Equipments. IEEE proceeding ARES '06: The First International Conference on Availability, Reliability and Security, Vienna, Austria, 2006. doi:10.1109/ARES.2006.71.

[13] R. McBride, Firewall Failover with pfsync and CARP, http://www.countersiege.com/doc/pfsync-carp/. Retrieved 2012/09/30.

[14] S. Goddard, R. Kieckhafer, and Y. Zhang, An Unavailability Analysis of Firewall Sandwich Configurations. *IEEE HASE '01*, 2001. doi:10.1109/HASE.2011.966815.

[15] S. Nadas and Ed. Ericsson, Virtual Router Redundancy Protocol (VRRP), RFC 5798, March 2010.

[16] T. Li, B. Cole et al. Cisco Hot Standby Router Protocol (HSRP), RFC 2281, March 1998.

[17] Y. Feng, Nen-Fu Huang, Rong-Tie Liu, and Meng-Huan Wu, Flow Digest: A State Replication Scheme for Stateful High Availability Cluster. *Proceedings of IEEE ICC2007, 2007.* doi:10.1109/ICC.2007.219.

[18] Y. Feng, Nen-Fu Huang, and Yen-Min Wu, Evaluation of TCP State Replication Methods for High-Availability Firewall Clusters, *Proceedings of IEEE GLOBECOM 2008, 2008.* doi:10.1109/GLOCOM.2008.ECP.389.

Kasom Koht-arsa received the Bachelor of Engineering and Master of Engineering degrees in computer engineering from Kasetsart University, Bangkok, Thailand, in 1999 and 2003, respectively.

He is currently a Network Engineer at the Faculty of Engineering, Kasetsart University. His research interests include Internet Security, Embedded System and High-speed Networking.

Surasak Sanguanpong received the Bachelor of Engineering and Master of Engineering degrees in electrical engineering from Kasetsart University, Bangkok, Thailand, in 1985 and 1987, respectively.

He is currently an Associate Professor at the Department of Computer Engineering, Faculty of Engineering, Kasetsart University. He is also the Director of the Applied Network Research Laboratory (ANRES), Kasetsart University. His researches focus on Network Operation and Management, Internet Security and High-speed Networking.

Assoc. Prof. Sanguanpong is a member of IEEE and ACM.