

The Raincore Distributed Session Service for Networking Elements

Chenggong Charles Fan

Jehoshua Bruck

California Institute of Technology
Mail-Stop 136-93
Pasadena, CA 91125
{fan, bruck}@paradise.caltech.edu

Abstract

Motivated by the explosive growth of the Internet, we study efficient and fault-tolerant distributed session layer protocols for networking elements. These protocols are designed to enable a network cluster to share the state information necessary for balancing network traffic and computation load among a group of networking elements. In addition, in the presence of failures, they allow network traffic to fail-over from failed networking elements to healthy ones. To maximize the overall network throughput of the networking cluster, we assume a unicast communication medium for these protocols.

The Raincore Distributed Session Service is based on a fault-tolerant token protocol, and provides group membership, reliable multicast and mutual exclusion services in a networking environment. We show that this service provides atomic reliable multicast with consistent ordering. We also show that Raincore token protocol consumes less overhead than a broadcast-based protocol in this environment in terms of CPU task-switching. The Raincore technology was transferred to Rainfinity, a startup company that is focusing on software for Internet reliability and performance. Rainwall, Rainfinity's first product, was developed using the Raincore Distributed Session Service. We present initial performance results of the Rainwall product that validates our design assumptions and goals.

1 Introduction

As Internet continues to grow, there emerges a strong need to eliminate single points of failures and to scale up performance bottlenecks that exist among the networking elements between the clients and the servers. Typically for a client to fetch information from a server on the Internet, the information must travel through multiple hops of network elements along the way. Routers, gateways, firewalls, proxy servers are examples of such devices (Figure 1). As millions of clients and servers are sending billions of bits through the Internet simultaneously, reliability and performance of these

devices naturally become key challenges facing the Internet infrastructure today.

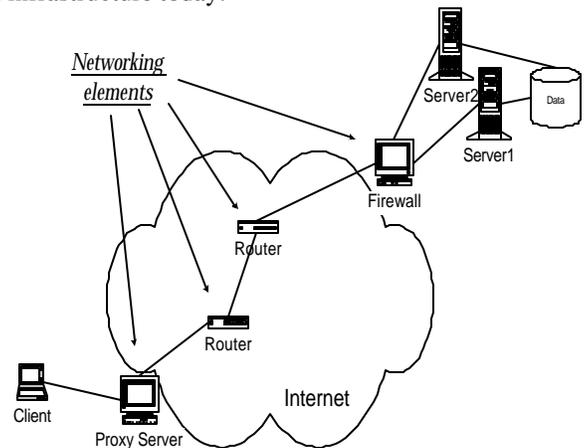


Figure 1. An example of a traffic path between a client and a server farm.

One promising way of improving the reliability and performance of the Internet is to cluster the networking elements. By having multiple networking elements work together for the same purpose, this in effect creates a distributed system in a networking environment. For example, in Figure 1, instead of having one firewall front-gate the server farm, a cluster of firewalls can be used. The objective of this distributed system is to enable both load balancing and fail-over among the member nodes. To load balance, there must be a way to distribute network traffic to members of the cluster, so that the overall throughput is multiplied. To fail over, the healthy nodes must discover nodes that has failed, and take over the networking traffic from the failed node, without interrupting the traffic flow.

As part of the RAIN project [Bohossian et al., 1999], we designed a set of distributed protocols that enable the clustering of networking elements in the environment described above. The overall architecture, the Raincore Distributed Services is described in Figure 2. It is our

hope that using Raincore, application developers will easily be able to create application solutions that run on top of a cluster of networking elements. The traffic load will be shared among the nodes in the cluster, and failures will not affect the availability of the overall network service.

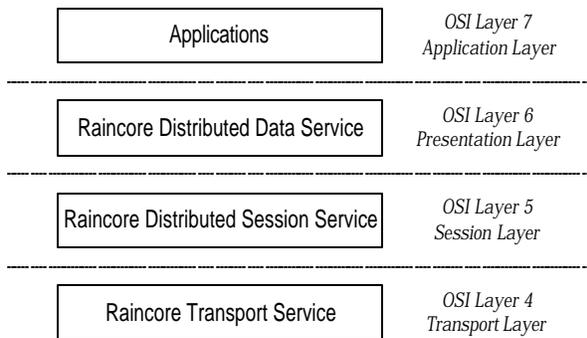


Figure 2. Raincore Distributed Services Architecture

As Figure 2 illustrates, the Raincore Distributed Services are mapped into Layer 4 through Layer 7 in the OSI 7-layer networking model [Tanenbaum, 1996]. While the advent of Internet is bringing together communication and computation, part of this convergence is reflected by this architecture that builds distributed computing protocols into the communication stack. We focus in this paper on the Raincore Distributed Session Service. It sits atop of the Raincore Transport Service, and uses a fault-tolerant token-ring protocol to provide group communication to the upper layers. In particular, it performs group membership management and reliable multicast with consistent ordering among the group members. In addition, it offers a mutual exclusion service that is useful for the Distributed Data Service, as well as the applications.

Group communication is a key component to enable a distributed system for the network environment, similar to its importance to any other distributed systems. It is a necessary module to maintain the group membership of the cluster, as well as to share state information among the member nodes. Load balancing and fail-over can take place based on the cluster membership and other critical cluster information shared by the group communication module. This module can also be used to share arbitrary application state, to facilitate transparent fail-over of traffic from a failed node to a healthy node, without the clients or the servers aware of the failures.

Group communication in a distributed system is a challenging topic that has been studied extensively, both

in theory [Fischer et al., 1985] [Birman and Joseph, 1987] [Moser et al., 1994] and in practice. The key functionalities that a group communication system must provide are group membership agreement and atomic reliable multicast with consistent ordering. A number of distributed systems projects have been implemented with group communication modules at their cores. Examples include the ISIS and HORUS projects [Birman and van Renesse, 1994] [van Renesse et al., 1994], the TRANSIS project [Amir et al., 1992], the TOTEM project [Amir et al., 1995], and the MPI project [Gropp et al., 1999]. A broadcast medium is assumed for these projects, and important concepts and novel algorithms have been invented and implemented for a broadcast environment to provide reliable multicast with multiple levels of consistency. These projects have greatly helped the advancement of both scientific distributed computing and parallel database implementations.

We focus our attention on the networking elements on the Internet. In this unique environment, the group communication is happening at the same time as the regular network traffic being processed by the members of the distributed system. Our design goal is then to have robust and consistent group communication among the member nodes, with minimal extra overhead on the CPU and the network. The main purpose of the group communication is to share among the networking devices the cluster state and the state information associated with the regular network traffic, so that load balancing and fail-over can occur smoothly. This will enable a cluster of networking elements to have the maximum throughput and the highest reliability. There are two implications from this unique property:

1. **Unicast-based Communication:** Unlike existing studies in distributed systems, broadcast-based medium can not be assumed. It is true that many of the currently popular local area networks are broadcast medium, example being an Ethernet environment using a hub. However, to be able to multicast messages to all members of the group, either each member set its network interface card to the promiscuous mode, or all the cards on all of the nodes must use a single multicast MAC address. This poses a limitation on the overall network traffic throughput of the networking cluster. Setting up a broadcast medium among the members of the cluster would mean that the total throughput of the entire cluster on that network is limited by the throughput of one network interface card, no matter how many nodes are in the cluster. In contrast, assuming a unicast model would allow the throughput of N network interface cards on a network, for a cluster of N nodes

interconnected by a switch. In this study, we make design decisions on a unicast-based network medium.

2. **CPU Task-switching Performance Metric:** In a networking environment, the primary job of the networking elements is to process the network traffic traveling through the device. Group communication plays an assisting role to organize the devices and share state for the devices in the cluster. The design goal of the group communication service is that it poses minimal additional CPU overhead and network overhead onto the cluster. Both the network overhead and the CPU overhead can be measured by the number of packets sent and received by each node in the cluster. A more subtle measurement of the CPU overhead is the number of task-switching actions that the CPU makes between the processing of regular network traffic, and the processing of the group communication. This measurement is important since the network elements are usually optimized to handle extremely high throughput, and switching between the traffic processing and group communication has significant latency cost.

These implications partly explained the reason of the gap between the state-of-art research in academia and the primitive standby solutions in the Internet industry today. In a standby solution, a secondary idle node is present to monitor the primary node, and will become active if the primary node fails. Being a useful high availability solution, this solution does not take advantage of the processing power of the secondary node, and offers no scalability.

In Section 2 of this paper we will give details about the token-ring protocol that is the core of the Raincore Distributed Session Service. In addition protocols that handles token loss and cluster partitioning and re-merge will be described in the same section. A number of applications have been implemented using the Raincore Distributed Session Service. In Section 3 we present how two of them. The first is the Virtual IP Manager developed by us at Caltech. This application ensures that a pool of Virtual IPs are always available to the outside world, in the presence of node failures. This application is used by a number of other applications, one of them being Rainwall, the second application introduced in this paper. Rainwall is a clustering solution for firewalls using Raincore that allows load balancing and transparent fail-over to happen among a number of firewall nodes. Rainwall is a shipping product from Rainfinity, a Caltech spin-off that provides performance

and reliability software for the Internet [<http://www.rainfinity.com>].

In Section 4 we compare the performance of the Raincore Distributed Session Service with broadcast based group communication protocols, and present the benchmark of an application developed using Raincore services. In Section 5 we conclude and state our future research directions.

2 Protocols

2.1 Raincore Transport Service

The Raincore Transport Service supply the atomic reliable unicast transport and failure-on-delivery notification to the Raincore Distributed Session Service. It is a module placed in the transport layer within the Raincore architecture. It requires the availability of an unreliable unicast interface to send and receive packets. In typical implementations, it uses UDP as the packet sending and receiving interface. Raincore Transport Service differentiates from TCP in three aspects:

1. It is an atomic point-to-point packet unicast mechanism with acknowledgement. A packet is either completely delivered, or not delivered at all. It does not have the concept of connections or streams. Therefore there is no connection state information to track as nodes go up and down.
2. The Transport Service allows each node to have multiple physical addresses. This allows redundant links between the nodes in the group, therefore makes the group more resilient to link failures and less likely being partitioned. Packet-sending strategy using multiple physical addresses can be specified, where the physical addresses can be targeted in sequential or parallel order.
3. The Transport Service generates notification to the upper layer both when it receives the acknowledgement from the destination, as well as when all sending efforts have failed. The second notification, the failure-on-delivery notification, is particularly significant, as it serves as a local-view failure detector for the Raincore Distributed Session Service.

2.2 The Token-Ring Protocol

At the core of the Raincore Distributed Session Protocol is a token-passing mechanism. Token ring is

one of the best-known protocols used in the distributed computing world. Started with the work by Chang and Maxemchuk [Chang and Maxemchuk 1984], and continued both in the ISIS/HORUS projects [Birman and van Renesse 1994] [van Renesse et al. 1994] and the TOTEM project [Amir et al. 1995], token ring have been used to sequence the broadcast message ordering and manage the group membership. In Raincore Distributed Session Service, the token ring protocol also serves as a “locomotive” for the reliable multicast transport. In other words, reliable multicast is achieved by piggybacking the messages to the token, while the token traverses the ring.

We will first describe the Raincore token-ring protocol in details. The nodes in the group are ordered in a logical ring. A TOKEN is a message that is being passed at a regular time interval from one node to the next node in the ring. The Raincore Transport Service is used for the transmission of the TOKEN, and provides the reliable unicast for the TOKEN. Contained in the TOKEN header is the authoritative knowledge of the group membership. When a node receives a TOKEN, it updates its local membership information according to the TOKEN, as well as modifies the information on the token based on its local information.

As the TOKEN is being passed around in the ring, each node operates as a state machine. When a node has a TOKEN, it is in the EATING state, when it does not have the TOKEN, it is in the HUNGRY state. During normal operation, a node is switching between the EATING state and the HUNGRY state at regular intervals, triggered by the TOKEN arrivals and departures. There is also a sequence number field on the TOKEN. Every time a TOKEN is being passed from one node to another, the sequence number on the TOKEN is increased by one.

The TOKEN is used for failure detection. Raincore uses an aggressive failure detection protocol that achieves fast failure detection convergence time. After a node fails to send a TOKEN to the next node, it receives a failure-on-delivery notification from the Transport Service. This node immediately decides that the target node has failed or disconnected, and removes that node from the membership. The node updates the TOKEN with the new membership information, and passes the TOKEN to the next healthy node in the membership ring.

The token mechanism is the basic component of the membership protocol. It guarantees that there exists no more than one TOKEN in the system at any one time. This single TOKEN maintained the authoritative group membership as it travels around the ring. The multicast messages are packed and attached to the TOKEN message, and travels the ring with it to reach every healthy node in the group. In addition, because of the uniqueness of the TOKEN, it serves as a master-lock to

make it possible to provide mutual exclusion services to the upper layers.

A few key questions still remain. What if a TOKEN is lost due to a node failure? How to add a new node to the system? How does the system recover from a transient failure? All of these questions can be answered by the 911 TOKEN-recovery and join protocol.

2.3 The 911 Token-recovery and Join Protocol

There is a timeout associated with the HUNGRY state. If a node remains in the HUNGRY state for a certain period of time, it enters the STARVING state. The node suspects that the TOKEN has been lost, and a special 911 message is sent to the next node in the ring. The 911 message is a request for the right to regenerate the TOKEN, and is to be approved by all the live nodes in the membership. If the TOKEN has not been lost, the 911 message will be denied by the node who owns the token. If indeed a TOKEN loss has occurred, it needs to be guaranteed that one and only one node should regenerate the TOKEN. To guarantee this mutual exclusivity, the sequence number on the TOKEN is used.

Each node makes a local copy of the TOKEN after each time the node receives it. When a node needs to send a 911 message to request the regeneration of the TOKEN, it puts on the 911 message the sequence number that is on their last local copy of the TOKEN. This sequence number will be compared to all the sequence numbers on the local copies of the TOKEN on the other live nodes. The 911 request will be denied by any node which possesses a local copy of a more recent TOKEN. In the case when the TOKEN is lost, every live node sends out a 911 request after their HUNGRY timeout expires. Only the node with the latest copy of TOKEN will be granted the right to regenerate the TOKEN.

The 911 message is not only used as a TOKEN regeneration request, but also as a request to join the membership. When a new node wishes to participate in the membership. It sends a 911 message to any node in the group. The receiving node notices that the originating node of

this 911 is not a member of the distributed system, and therefore treats it as a join request. The next time it gets the TOKEN, if the new node has not yet been added to the membership, it will add it to the membership on the token. It then sends the TOKEN to the new node. The new node thus becomes a part of the system.

The unification of the TOKEN regeneration request and the join request facilitates the treatment of the link failures in aggressive failure detection protocol. For example, in the group ABCD, the link between A and B fails. Node B is removed from the membership and the

ring becomes ACD. Node B stayed in HUNGRY mode for a while and then enters the STARVING mode. It sends out a 911 message to node C according to the protocol. Node C notices that node B is not part of the membership and therefore treats the 911 as a join request. The ring is then changed to ACBD. Not only does Node B rejoin the membership, but the broken link is also naturally bypassed in the new ring.

With the same mechanism, failure detector false alarms can be corrected. When a false alarm occurs, a node was removed from the membership by mistake. It sends out an 911 as it enters the STARVING state. The 911 message is recognized as a join request, and the node is being added back into the group. This shows that while the failure detector can never be 100% correct, after a wrong decision is made, the wrongfully excluded node will be able to automatically rejoin the group given the 911 protocol.

2.4 The Split-brain and the Group Merge Protocol

The partitioning of a group is a well-known problem in this field for which there exists no perfect solution. This problem arises when the group is broken into more than one sub-groups. Each sub-group by itself is functional, but it cannot communicate with other sub-groups. This problem is also referred to as the “split brain” problem. There are two common strategies to this problem. The first strategy is a quorum decider. If N is the maximum size of the group, when the size of the group is $N/2$ or less, every node in the group shuts down itself. This is a safe strategy that prevents the split brain from happening. However, it sets severe limitations on the scalability and fault-tolerance capabilities. The second strategy is to allow each partitioned sub-group continuing to be functional, and design a protocol to discover when the communication between sub-groups resumes, and to merge the subgroups into one group.

In the Raincore design, while a pure quorum decider is not used, attention has been paid to first prevent the brain from being splited. The Raincore Transport Service supports redundant communication links between nodes, which makes the isolation of sub-groups less likely to occur. Another feature that Raincore offers is the ability to define critical resources for each of the member nodes. A node will shut down itself when any of its critical resources becomes unavailable. Sometimes it makes sense in a network environment to use this mechanism to setup a common critical resource, such as an Internet connection, for a group of nodes. When the group partitions, only one sub-group will continue to have the connectivity to that common resource, and all other sub-groups will shut down themselves, thereby preventing the split-brain. This strategy is not always

attractive, since it does introduce that common resource as a single point of failure.

When all preventions fail, Raincore will allow all sub-groups to continue to function on their own. When the communication between sub-groups resumes, it is desirable to merge the sub-groups. In order for the merge to happen, the Raincore session layer protocol suite contains two more protocols, the Raincore Discovery Protocol and the Raincore Merge Protocol. The Discovery Protocol is for the sub-groups to discover the existence of each other. The Merge Protocol is designed to perform the merging between the sub-groups.

To explain the Raincore Discovery Protocol, we first introduce the concept of Eligible Membership. The Eligible Membership contains the IDs of all eligible members of the group. It is configured on each member node, and the configuration can be changed and updated online. Each healthy member node sends a BODYODOR message periodically to the other nodes who are in the Eligible Membership, but not the current Group Membership. BODYODOR message is a small message sent with a regular, but low frequency, so that it does not impose a major overhead onto the system. Contained in the BODYODOR message is the node ID of the sender, and the group ID of sender's current group. It is common to use the lowest node ID in the current Group Membership as the group ID. When a node receives a BODYODOR message, and if the sender is not in its own membership, but is in its Eligible Membership, the receiver has discovered a new eligible node. The goal of the Discovery Protocol is achieved.

Now the challenge is to merge the TOKENs. In order to avoid deadlocks in the merge process, the group IDs are used as tie-breakers. The BODYODOR message is considered as a join request if and only if the sender's group ID is lower than the receiver's group ID. When it is regarded as a join message, the receiving node will wait for its TOKEN, check that the BODYODOR sender is not on the membership on the TOKEN, add the BODYODOR sender to the membership, mark the special TBM (To Be Merged) flag on the TOKEN, and send the TBM TOKEN to the BODYODOR sender.

When the BODYODOR sender receives a TBM TOKEN, it will wait for its original token, merge the memberships on the two tokens, and concatenated the multicast messages attached to the two tokens, and merge the two tokens into one. Thereby the merge between the two sub-groups are completed. When there are more than two subgroups, by using the group ID ordering, the eventual merge among all of them can be completed without deadlocks.

2.5 Group Membership

It has been shown that it is impossible to achieve consensus on group membership in asynchronous environment [Fischer, et al., 1985] [Chandra, et al., 1996]. The root of the impossibility lies in the lack of a reliable failure detector: one cannot distinguish between a failed node and a slow node. Recent research, however, has been exploring the new ways of looking at the group membership problems, so that properties of different group membership protocols can be proved [Neiger, 1996] [Franceschetti, 1999].

We define the Quiescent Period to illustrate the group membership property that the Raincore protocol satisfies. There are two possible group membership change events: A node-removal event triggered by a node which failed or voluntarily left; and a node-addition event triggered by a node recovery or join or a false-alarm in the failure detector. In an asynchronous environment, if these events happen at arbitrary frequency, it is impossible to prove the correctness of any group membership protocol.

A Quiescent Period is a period when none of these three events occur. It is then possible to show that the agreement on group membership can be achieved during the Quiescent Period which lasts long enough. First, we show the uniqueness of the TOKEN -- that at no time there are more than one TOKEN existed in a group. Second, we show the everlasting of the TOKEN that when a TOKEN disappears from the system due to node failure, it will be regenerated within a finite amount of time. As failures, joins and failure-detector false-alarms happen at arbitrary frequency, given the uniqueness and everlasting of the TOKEN, there is a unique authoritative membership in the group, which is recorded on the TOKEN. In the Quiescent Period, the TOKEN will be able to traverse all nodes in the authoritative membership, copy the authoritative membership to every node in the group. Since during the Quiescence Period there is no change to the authoritative membership, the group membership agreement among all member nodes is achieved.

2.6 Reliable Multicast

Reliable Multicast allows one node to communicate to a group of nodes in a reliable manner. It is the primary service that a group communication module should provide. In addition to the reliable point to point delivery that can be accomplished by an acknowledgement scheme, there are two additional properties that are desired from a Reliable Multicast service: atomicity and consistent ordering [Tanenbaum, 1995]. Atomicity refers to the all-or-nothing property, that when a message is sent to a group of nodes, it will correctly arrive at either all members of the group, or none of them. Consistent Ordering refers to the desire

that all nodes in the group should receive all messages in the same order.

Raincore Distributed Session Service directly provides closed group communication. A member of the group can use Raincore to reliably multicast to all other members of the group. Raincore achieves both atomicity and consistent ordering for closed group communication. In addition, open group communication between a node outside the Raincore group and the Raincore group can be achieved. A node can send a message to any member of the Raincore group, and that member then forwards the message to the entire group using Raincore.

As we discussed in Section 2.1, the messages are packed and attached to the TOKEN, and circulated around the ring, until it comes back to the originating node. The TOKEN together with the messages are transported using a reliable point-to-point transport service. When the TOKEN returns to the originating nodes, all nodes on the membership have received all of the attached message from this originating nodes, and therefore atomicity is guaranteed.

Atomic multicast can be achieved with three different levels of consistent ordering [Amir et al., 1992]. Causal ordering guarantees all receiving nodes agree on the order of messages that are causally related to all nodes; agreed ordering guarantees that all receiving nodes agree on the ordering of all messages.; safe ordering guarantees that each node delivers a message to upper layer only after all nodes in the membership have acknowledged the reception.

The first two levels of consistent ordering are naturally achieved by the Raincore token-ring protocol. Because of the uniqueness of the token, the single delivery locomotive for the multicast, the message ordering on the token decides the message ordering on each of the node, thereby guarantee the agreement. Interestingly, it requires no extra cost to achieve agreed ordering than no ordering. Safe multicast can also be achieved by Raincore, which requires that TOKEN travels one more round, to guarantee the receipt by all members before the Raincore Distributed Session Service passing the message to the upper layer.

2.7 Mutual Exclusion

The Raincore Distributed Session Service includes a mutual exclusion service as well. Because of the uniqueness of the TOKEN, it guarantees that at most one node can be in the EATING state at any time. Because the TOKEN is circulating around in the ring, each node has a fair chance of getting the TOKEN. The 911 protocol makes this token-based mutual exclusion fault-tolerant. When a node is in the EATING state, it is assured that no other node is EATING, and that its change to global data is authoritative.

This is a powerful service for the upper layers. In particular, a Raincore distributed lock manager is implemented as part of the Raincore Distributed Data Service, using the mutual exclusion service to acquire and release data locks. The data locks provided by the distributed lock manager, comparing to this master-lock, can be associated with one or more shared data items, and can be owned by a node without requiring the node to remain in the EATING state.

3 Applications

3.1 Virtual IP manager

One way of distributing traffic to a group of networking elements is by maintaining a pool of highly available virtual IPs among the group members. These virtual IPs are the actual publicly advertised IP addresses for this network cluster. All traffic that goes through the cluster is being directed to one of the virtual IPs. The virtual IPs are mutually exclusively assigned to different nodes in the cluster by the Virtual IP manager. In the presence of failures, Raincore Distributed Session Service discovers the failure and the Virtual IP manager promptly moves all the virtual IPs that was owned by the failed node to healthy ones. The Virtual IPs can also be moved for load balancing or other reasons.

When a virtual IP is being moved from one node to another, a gratuitous ARP is being sent to refresh the ARP cache of all computers and routers on that subnet. While the virtual IPs are being moved around, MAC addresses are never moved and remain unique to each node. Therefore, the virtual IP manager assures users that while physical machines can go down, the virtual IPs never disappear as long as at least one physical node is functional.

The Virtual IP Manager uses the Raincore services to share the assignment of the virtual IPs, as well as uses the master-lock to make sure that there is no conflict in the virtual IP address assignments. This module is an important building block in using distributed computing technology to provide load balancing and fail-over for networking devices. After this technology is transferred to Rainfinity, a number of applications are created based on it. Rainwall is one example.

3.2 Rainwall

Firewall is essentially a router that filters traffic according to a security policy. It is popular at the entry points of an enterprise network. Rainwall is a commercial application using Raincore Distributed Services to deliver a high-availability and load-balancing clustering solution for firewalls. It allows firewall not to

become a single point-of-failure or a performance bottleneck for the customers.

In addition to the Virtual IP Manager that provides coarse load balancing and traffic fail-over among the firewalls, Rainwall also includes a kernel-level software packet engine that load-balances traffic connection by connection to all firewall nodes in the cluster. The load and connection assignment information are shared among the cluster using the Raincore Distributed Session Service.

In fact Rainwall not only works with firewalls, but also works in general with any routers or gateways. It creates a cluster of virtual routers that never fails, as long as at least one physical router is operational. It also allows a variety of applications to be present on the routers and is able to monitor the health of critical resources such as the applications, the network interfaces, as well as the remote Internet links. When any of the critical resource fails, Rainwall will shift traffic away from the failed node.

The fail-over time of Rainwall is under two seconds. For example, suppose a client is downloading a file from a server through a firewall. If a network cable connecting one of the Rainwall firewalls is accidentally unplugged, the client, instead of losing the connection, will only see about 2-seconds hick-up in the traffic flow, before it fully resumes. Because of these convincing capabilities, Rainwall has been well received by the customers and is operational at more than 100 major customer sites worldwide to provide high availability and load balancing to their firewalls.

4 Performance

4.1 Overhead Analysis

There remains the question whether letting TOKEN to be the multicast transport has satisfactory performance to validate it as a practical design. In particular, we would like to answer how the performance of Raincore compares to the broadcast based protocols, in terms of CPU overhead, network overhead and message delivery latency. To address this fully, we need to evaluate the environment very closely. Raincore is designed for a high throughput, high-speed networking environment. It is realistic to assume that the network latency is very low. This fact alleviates the latency concerns over the token-based protocols.

On the other hand, huge amount of network traffic are passing through the cluster at the same time Raincore is used for intra-cluster state sharing. In addition, the CPUs in the networking elements are frequently

optimized for high throughput single-task processing of network traffic, where task-switching can be costly. Raincore is particularly attractive in terms of number of task-switching actions required from the CPU.

For example, suppose a cluster contains N networking devices, and each device needs to send M messages to the group every seconds. And suppose that the TOKEN travel around the cluster at a frequency of L roundtrips per second, $L < M$. Using Raincore, only L task-switching actions are needed every second to achieve reliable atomic consistent multicast. Using a broadcast-based protocol, at least $M \times N$ task-switching actions are needed. If a two-phase commit protocol is used to guarantee consistent ordering, up to $6 \times M \times N$ task-switching actions are needed at every node. This will impact both the CPU overhead, and the latency on the group communication and the processing of the network traffic.

Now let us look at the network overhead. While it is true network mediums such as Ethernet is in its nature a broadcast medium. To configure it to be one in the real-life have real penalties. Either all nodes need to share the same MAC address, or they need to be configured to work in promiscuous mode. Both methods impose undesirable performance bottleneck for the processing of regular network traffic. For example, in a Fast Ethernet environment, to configure the cluster to share a broadcast medium means that no more than 100 Mbps can travel through the cluster of N nodes in any direction. In contrast, in a switched unicast Fast Ethernet environment, the aggregate throughput of the cluster can reach $N \times 100$ Mbps.

In a unicast environment, broadcast messages are achieved by sending multiple unicast messages, which could introduce more overhead than a token-ring based protocol. For example, in a cluster of N nodes, when each node needs to multicast one message of M bytes, there will be $(N-1)^2$ packets of M bytes on the network when a broadcast-based protocol is used. Number of packets will be doubled if acknowledgements are implemented for reliable delivery. The packet count will be further increased when consistent ordering is required. In contrast, using the token-based protocol, there are N packets of $N \times M$ bytes on the network, and the delivery is reliable and the order of delivery is consistent.

4.2 Rainwall Benchmark

A number of benchmark tests have been performed on Rainwall, both to understand the scaling capability of the Rainwall application and to validate the approach of the Raincore services. The Benchmark presented in Figure 3 is obtained from the Rainfinity Lab. The results were obtained by running Rainwall on a cluster of one, two and four gateways. The throughput number on the

chart indicates the total web traffic that travels through the Rainwall cluster. We can see that the throughput scaling from 1 node to 2 nodes is 1.97, and that the scaling from 1 node to 4 nodes is 3.76. The low overhead of the Raincore Distributed Services contributes to this impressive scaling. Throughout the test, Rainwall CPU usage is below 1%.

The benchmark test was performed using Sun Ultra-5 single-CPU 360MHz workstations as the Rainwall gateways in a Fast Ethernet switched environment. Each workstation has 256 MB of RAM and runs Solaris 2.6. HTTP clients are placed at one side to request data from Apache web servers on the other side of the Rainwall cluster.

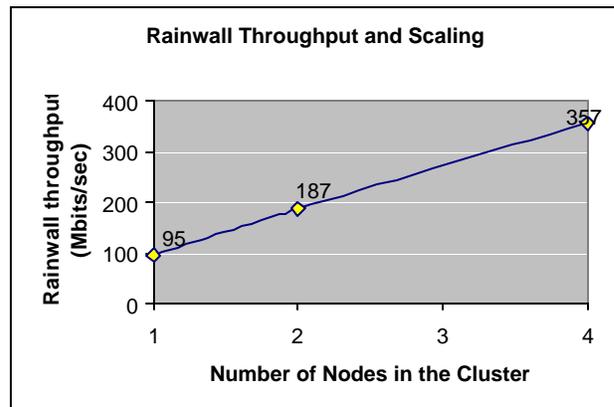


Figure 3. Rainwall Performance Benchmark

5 Conclusions and Future Work

Internet is bringing communication and computation together like never before. This is an opportunity for us to work on the marriage between the distributed computing protocols and the networking protocols. Raincore is an attempt in that direction, striving for a set of practical and efficient distributed services for networking elements. Raincore Distributed Session Service is at the heart of this suite of services. It has the design goal of minimizing its impact on the overall throughput of the network cluster.

There are three efforts in the research and development of Raincore Distributed Services happening right now and in the near future:

- The Group Communication Protocols are being extended to address more challenging scenarios. For example, we are currently working on the hierarchical design that extends the scalability of the protocol.
- The development of the Raincore Distributed Data Service is underway. The ambition is to

provide developers an environment where they will be able to develop distributed networking applications with the ease of developing a multi-thread shared-memory application on a single processor.

- Further work is underway to build the Raincore Distributed Services more deeply into the networking stack, and integrate more tightly with the packet engine, thereby creating a powerful distributed general-purpose packet engine.

Acknowledgement

The authors would like to thank Cheng Tan, Paolo Carnevali, Jie Yu, Paul LeMahieu, Phil Love and Vincent Bohossian for their valuable contributions to the Raincore architecture and implementation.

References

- Amir, Y., Moser, L. E., Melliar-Smith, P. M., Agarwal, D. A., Ciarfella, P. 1995. The Totem Single-Ring Ordering and Membership Protocol. In *ACM Transactions On Computer Systems*, 13(4), pages 311-342, November 1995.
- Amir, Y., Dolev, D., Kramer, S., and Malki, D. 1992. Transis: A communication sub-system for high availability. In *Proceedings of the IEEE 22nd Annual International Symposium on Fault-Tolerant Computing* (Boston, MA). IEEE, New York, 76-84.
- Birman, K. P. and Renesse, R. V. 1994. *Reliable Distributed Computing with the ISIS Toolkit*. IEEE Computer Society Press.
- Birman, K. P. and Joseph, T. A., 1987. Reliable communication in the presence of failures. *ACM Transaction on Computer Systems*, 5(1), pages 47-76, February 1987.
- Bohossian, V., Fan, C. C., LeMahieu, P. S., Riedel, M. D., Xu, L., and Bruck, J. 1999. Computing in the RAIN: Reliable Array of Independent Nodes. *Caltech Paradise Electronic Technical Report ETR029*, 1999.
- Chandra, T.D., Hadzillacos, V., Toueg, S. and Charron-Bost, B., 1996. On the impossibility of group membership. In *Proceedings of the fifteenth ACM symposium on principles of Distributed Computing*, ACM Press, 322-330.
- Chang, J. M. and Maxemchuk, N. F. 1984. Reliable broadcast protocols. *ACM Transactions on Computers Systems* 2, 3 (August), 251-273.
- Fischer, M. J., Lynch, N. A., and Paterson, M. S. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM* 32, 2 (April), 374-382.
- Franceschetti, M., and J. Bruck, 1999. On the Possibility of Group Membership. *Dependable Network Computing*, Kluwer, Nov. 1999.
- Gropp, W., Lusk E. and Skjellum, J., 1999. *Using Mpi : Portable Parallel Programming With the Message-Passing Interface*. MIT Press, 1998.
- Moser, L. E., Melliar-Smith, P. M., and Agrawala, V. 1994. Processor membership in asynchronous distributed systems. *IEEE Transactions on Parallel and Distributed Systems* 5, 5 (May), 459-473.
- Neiger, G., 1996. A New Look at Membership Services. In *Proceedings of the Fifteenth ACM Symposium on Principles of Distributed Computing*, ACM press, 331-340.
- Ricciardi, A. M., and Birman, K. P., 1991. Using Process Groups to Implement Failure Detection in Asynchronous Environments. In *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, ACM Press, 341-352, May 1991.
- Tanenbaum, A. S., 1995. *Distributed Operating Systems*. Prentice-Hall, New Jersey, 1995.
- Tanenbaum, A. S., 1996. *Computer Networks, third edition*. Prentice-Hall, New Jersey, 1996.
- Van Renesse, R., Hickey, T. M., and Birman, K. P. 1994. Design and performance of Horus: A lightweight group communications system. Technical Report 94-1441 (August), Cornell University, Department of Computer Science.