Report Number: WUCSE-2011-101

2011

The Forest Overlay Network

Logan Stafman

Forest is an overlay network designed for large real-time distributed systems. In particular, we're interested in virtual worlds that provide high-quality interaction over a constantly changing pattern of communication. Forest is suitable for applications in which many entities send data to a large set of constantly changing entities. By using tree-structured channels, we can create logically isolated private networks which support both unicast and multicast routing. In this paper, we will discuss the core components of Forest and measure the performance of the control elements of the network. We will also provide examples of control sequences and the roles... **Read complete abstract on page 2.**

The Forest Overlay Network

Logan Stafman

Complete Abstract:

Forest is an overlay network designed for large real-time distributed systems. In particular, we're interested in virtual worlds that provide high-quality interaction over a constantly changing pattern of communication. Forest is suitable for applications in which many entities send data to a large set of constantly changing entities. By using tree-structured channels, we can create logically isolated private networks which support both unicast and multicast routing. In this paper, we will discuss the core components of Forest and measure the performance of the control elements of the network. We will also provide examples of control sequences and the roles played by control elements in those sequences to help maintain fast, reliable data delivery.

2011-101

The Forest Overlay Network

Authors: Logan Stafman

Corresponding Author: lstafman@wustl.edu

Abstract: Forest is an overlay network designed for large real-time distributed systems. In particular, we're interested in virtual worlds that provide high-quality interaction over a constantly changing pattern of communication. Forest is suitable for applications in which many entities send data to a large set of constantly changing entities. By using tree-structured channels, we can create logically isolated private networks which support both unicast and multicast routing. In this paper, we will discuss the core components of Forest and measure the performance of the control elements of the network. We will also provide examples of control sequences and the roles played by control elements in those sequences to help maintain fast, reliable data delivery.

Type of Report: Other

Introduction

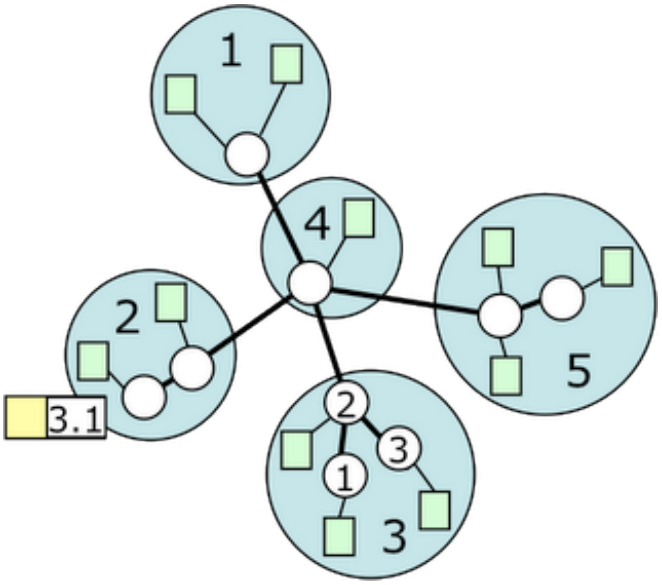
Forest is an overlay network designed for large distributed real-time applications such as online virtual worlds. In order to benefit from Forest, an application must involve many entities sharing real-time status with a constantly changing subset of other entities. The status updates issued by clients can be sent up to tens of times per second. In order to keep high-quality interaction, these packets must be delivered non-stop even as communication patterns change.

A tree-structured channel in Forest is called a comtree, or communications tree. This channel supports both unicast and multicast packet delivery. Each comtree can be viewed as a private network that is logically isolated and can be provisioned to meet bandwidth requirements of the application. For example, in the case of a virtual world, each comtree might be a separate world; clients from one world cannot interact with or affect the worlds of clients in other worlds. Clients can join and leave comtrees easily by sending requests to the *Comtree Controller*.

Forest routers use tables to manage their links, routes, and comtree links, and interfaces. For example, when a client (or, in the comtree sense, leaf) joins the network, the link table of the router must be updated to include the new link. When a client joins a data comtree, the comtree link table is updated accordingly. The routers together keep a global view of the *topology* of the Forest network.

Addressing in Forest

Forest addresses are 32 bit numbers. The high order bit distinguishes between unicast and multicast addresses; a 1 here identifies this address as a multicast address. Unicast addresses have two parts, the *zip code* and the *local address*. The zip code is 15 bits long and the local address is 16 bits long, allowing 2.1 billion addresses on a single comtree. Note that addresses in different comtrees are independent of one another, so we have many more unicast addresses available.

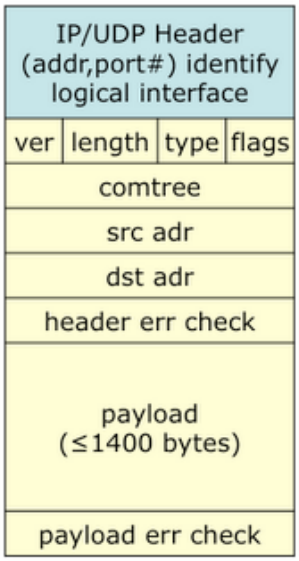


Unicast addresses that are in the same zip code must form a single connected subtree of the overall comtree in order to keep routing scalable. This means that from any node outside of a zip code, there must exist a single link by which nodes in that zip code can be reached. With this restriction, routers can maintain a single route for foreign zip codes, allowing local routers to forward packets to local endpoints. In the example above, an endpoint in zip code 2 is attempting to send a packet to an endpoint in zip code 3. Due to the fact that zip code 3 is a subtree, there exists one link from zip code 2 to zip code 3.

Packet Format

Packets on the Forest network have some additional information to allow for quick routing. Below is a description of a forest packet.

- *Version number*. First four bits – currently at 1.
- *Packet Length.* Specified in bytes, the field is 12 bits long. 1400 is maximum size.
- *Flags*. This is an 8 bit field. Currently, the only flag is the route request flag set by a router to request an endpoint.
- *Comtree.* This is a 32 bit field which identifies the comtree on which this packet should be processed.
- *Source Address*. This 32 bit field specifies the unicast forest address of the sender.
- *Destination Address.* This 32 bit field specifies the unicast or multicast forest address of the receiver.
- *Header Error Check*. This is a 32 bit check sum for the header.
- *Payload Error Check.* This is a 32 bit check sum for the payload (excluding the header).



Topology File Format

Each Forest Network has a specific *topology* that must be described in a topology file. This topology outlines everything about the instance of Forest, including routers, their interfaces, predefined links to control elements, and definitions for all of the predefined comtrees (though comtrees can be added dynamically as well). Below is a description of the topology file format.

Routers: The router section contains each router, the range of legal Forest addresses it can assign to clients, its physical location (in order to select the closest router to a client), its Forest Address, and a list of its interfaces. Each interface includes an IP address, the links that interface is in charge of, and maximum bit rates and packet rates allowed on that interface.

Leaf Nodes: While clients can connect dynamically, the links for the control elements must be predefined in the topology. Each leaf node has a name, an IP address, a Forest address, and a location.

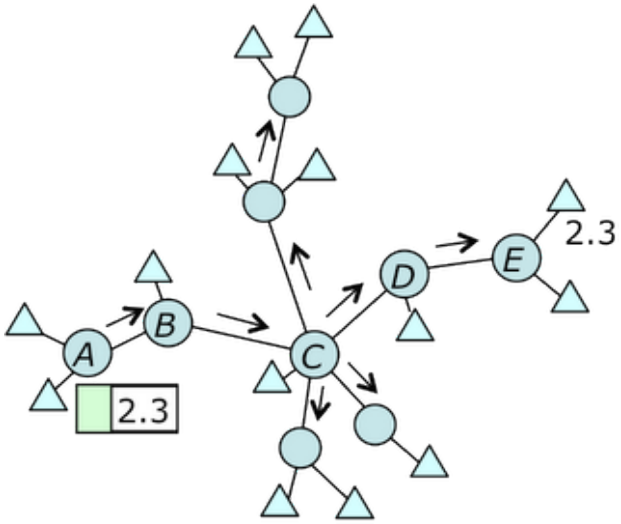
Links: Any backbone links (e.g. links connecting routers) should be defined in this section. In addition, links to predefined leaf nodes should be listed here. Each link has the two nodes it connects, its length, its maximum packet rate, and its maximum bitrate.

Comtrees: This section contains a listing of each of the comtrees in this instance of Forest. Both data and control comtrees should be defined here. Each comtree should have a comtree number, a root node, and the upstream and downstream packet rates for backbones and for leaf links. In addition, all predefined comtree links should be defined here; each set of comtree links must be a subset of the links previously defined in the links section. Links to controllers must be present on control comtrees.

Packet Forwarding

Unicast Forwarding

Each router typically holds routes for each foreign zip (e.g. the first half of the Forest address), as well as routes to all local addresses. These routes to foreign addresses are acquired using a self-learning process similar to that used in Ethernet LANs. If a router receives a packet for which it has no route, it will forward the packet on each outgoing link and set the *Route Request Flag*, asking any endpoint that already has a route to the destination address to respond with a *Route Reply Packet.*

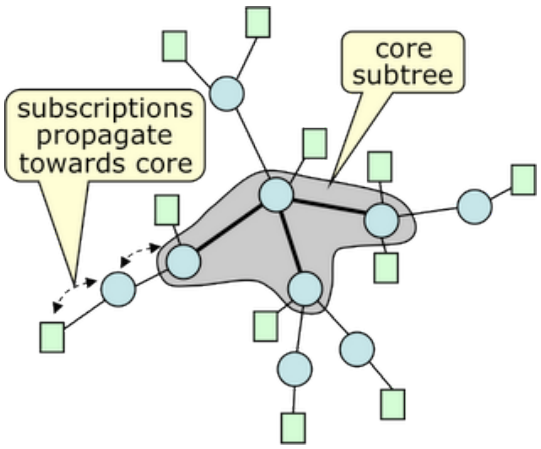


Above, a client on the left wants a route to Forest address 2.3. Assuming router E has a route to 2.3, the packet is flooded across the Forest network, until router E replies with a *Route Reply packet* and turn off the *Route Request* flag. If an earlier router in the path (e.g. router C) has a route to 2.3 already, then the network will not be flooded, and C will simply respond with the route to the destination. Local addresses are known to access routers when an endpoint is added to the Forest Network.

Routers have the option, but are not required, to process *Route Request packets* by storing the route to the requesting router. For example, if router C does not have a route to router A, C could choose to add that route when it receives the *Route Request packet.* However, this is not recommended as this adds unnecessary overhead to normal packet processing.

Multicast Forwarding

Each comtree in Forest has its own set of multicast addresses. Clients subscribe to multicast groups by sending *Multicast subscription* packets. It is the job of the routers to maintain these multicast groups and forward them accordingly to the *multicast core*. The core is a subset of nodes that **must** include the root node of the comtree. In the simplest case, the core is simply the root node.



The core may extend beyond the root node. Multicast packets must be forwarded only to the "edge" of the core, as all core nodes keep track of multicast groups. When the core is smaller, bandwidth is minimized, while a larger core minimizes subscription processing overhead and response time.

Capacity Provisioning of Comtrees

Clients in Forest can issue data reports using multicast as often as they like. By allowing capacity provisioning of comtrees, we ensure that all endpoints ensure reliable and quick data delivery, even as multicast subscriptions change.

Forest provides two options for capacity provisioning. The *Uniform Provisioning Option* assigns bandwidth to comtree links based on four different rates: *Upstream Backbone Rate, Downstream Backbone Rate, Upstream Access Rate,* and *Downstream Access Rate*. Backbone links are those connecting pairs of routers, while access links are those connecting leaf nodes to access routers. Upstream refers to packets travelling toward the root node, while downstream refers to traffic travelling away from the root node. Typically, downstream rates will be larger than upstream rates.

The *Autoprovisioning Option* configures the rates automatically based on access link rates. The rates are chosen in order to guarantee that all leaf nodes can send at their maximum upstream rates and receive at maximum downstream rates. These rates are computed using a simple tree traversal.

Control Elements

Throughout the Forest Network, there are several different activities that require control packets or sequences of control packets in order to be executed correctly. These include

- A client joining the Forest network
- A router joining the Forest network
- A client sending a *client connect* packet, and the resulting forwarding of that packet
- A client subscribing or unsubscribing

By using these control elements, we can translate client desires into appropriate changes in the topology of the Forest network.

1. Client Manager

The Client Manager has a number of responsibilities: Verifying usernames and passwords given by the clients, requesting that the network manager select an access router and configure the link on that router, informing the client of its Forest address, and making entries to a log which a client connects or disconnects.

The Client Manager acts primarily as a login server for avatars. Upon starting up, the avatars are not a part of the Forest network, and must be added. They create a TCP connection with the client manager, giving it their IP and port. Having a pool of threads, the Client Manager begins to use an unused thread, which guides the client through the process of joining the Forest network. The thread sends a *new client* control packet to the Network Manager, and receives a reply with the Forest address of the client and the router it is to connect to. The Client Manager then hands this information to the client, or a -1 if the Network Manager could not successfully add the client to the Forest network. Currently this router is determined via IP masking, but will be based on geographic location. In the future, a web interface will allow users to make accounts and start their Forest sessions. The Client Manager is also in charge of documenting *client connect* and *client disconnect* packets received to ensure that every initialized client connects and disconnects from Forest appropriately.

2. Network Manager

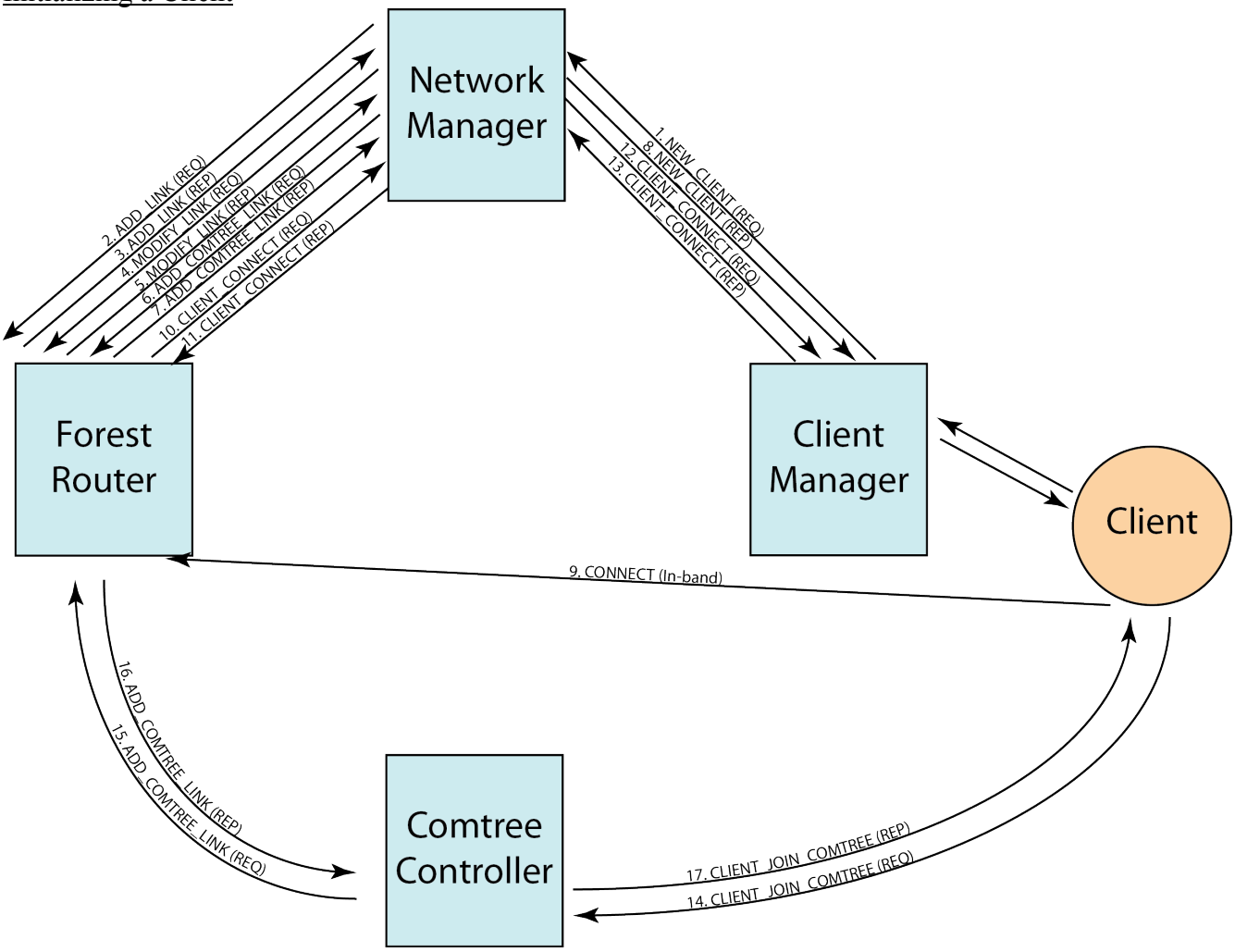
The Network Manager is in charge of both starting the Forest routers and keeping charge over routers and links in the Forest network. When the Network Manager receives a *new client*, *client*

connect/disconnect, or *boot request* packet, the Network Manager uses an unused thread to guide each event through the appropriate process. The Network Manager requires a *topology file* of the Forest Network in order to correctly boot routers and clients. It also allows administrators to manipulate the Forest network via an external connection to the *Console*. This console allows Forest network administrators to edit the Forest network in real time (e.g. adding/dropping links).

3. Comtree Controller

The Comtree Controller handles requests for clients to switch comtrees. Once the Network Manager has added the client to the control comtree, it is necessary for a client to join a data comtree in order to interact with other avatars. The client will send a "client join comtree" request to the Comtree Controller, which will send an add comtree link request to the router, including the data comtree and the avatar to be added. Upon success, the comtree controller will issue a modify comtree request to change bit rates and packet rates as specified by the topology file. As an avatar decides to switch among the data comtrees, this process will be repeated. The Comtree Controller also hosts an external connection to the *Comtree Display*, which allows administrators to view a real-time representation of the separate comtrees, and determine how many clients are currently present at each router on each comtree.

Initializing a Client



When a client is added to the Forest Network, it follows these steps:

1. The Client Manager sends a NEW_CLIENT request to the Network Manager, telling it the IP address and port of the client.
2. The Network Manager decides which router the client should connect to, and sends an ADD_LINK request.
3. The Router replies to the Network Manager with a link number.
4. The Network Manager requests that the Router adjust the bit rate and packet rate of the new link to an appropriate amount for a client with a MODIFY_LINK request.
5. The router replies to the Network Manager with a MODIFY_LINK reply.
6. The Network Manager sends an ADD_COMTREE_LINK request for the client signaling comtree. If there is more than one, this step is repeated.
7. The Router replies with an ADD_COMTREE_LINK reply.
8. The Network Manager replies to the Client Manager with a NEW_CLIENT reply, containing the Forest address, the Router's IP, and the Router's Forest address.
9. The Client sends an in-band connect packet to the router.
10. The router sends a CLIENT_CONNECT request to the Network Manager, notifying it that a client connected.
11. The Network Manager confirms that it received the router's packet.
12. The Network Manager forwards the CLIENT_CONNECT packet to the Client Manager.

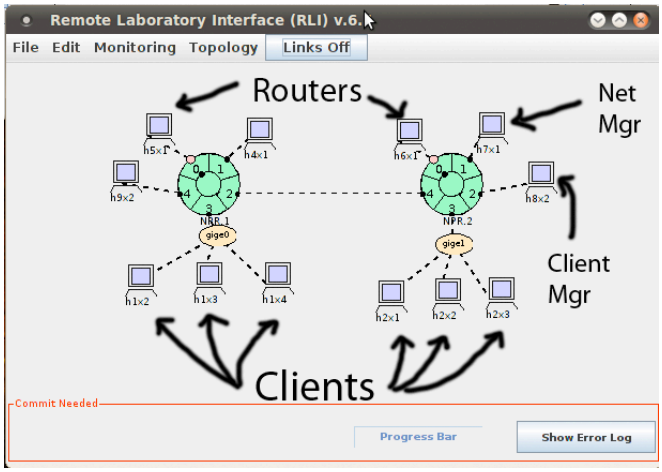
13. The Client Manager logs this packet and acknowledges the Network Manager.
14. The client sends a CLIENT_JOIN_COMTREE request to the Comtree Controller, to add itself to the appropriate data comtree.
15. The Comtree Controller sends an ADD_COMTREE_LINK request to the router.
16. The router sends an ADD_COMTREE_LINK reply to the Comtree Controller.
17. The Comtree Controller sends a positive CLIENT_JOIN_COMTREE reply to the client.

Initializing a Router

When a Router wants to join the Forest network, it sends a boot request control packet to the Network Manager. The Network Manager replies with the information about that router as specified in the topology file.

1. The router sends a BOOT_REQUEST packet to the Network Manager.
2. The Network Manager decides which router this is based on its IP address, and sends a BOOT_REQUEST reply.
3. The Network Manager sends an ADD_IFACE request for each interface defined in the *topology*. If a negative reply is received, a BOOT_ABORT message is sent.
4. The Network Manager sends an ADD_LINK request for each link defined in the *topology*. If a negative reply is received, a BOOT_ABORT message is sent.
5. The Network Manager sends a MODIFY_LINK request for each link that was just added. If a negative reply is received, a BOOT_ABORT message is sent.
6. The Network Manager sends an ADD_COMTREE request for each comtree specified by the *topology*. If a negative reply is received, a BOOT_ABORT message is sent.
7. The Network Manager sends an ADD_COMTREE_LINK request for each comtree link in the *topology*. If a negative reply is received, a BOOT_ABORT message is sent.

Control Element performance

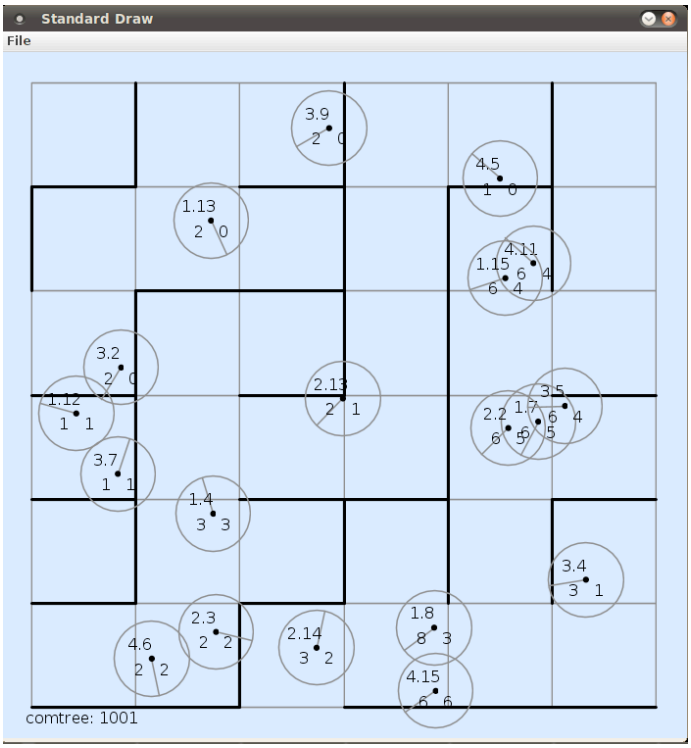


The speeds of these control elements are the limiting factor for the overall speed of the Forest Network. In order to fully test their speeds, an experiment was devised which put strain on the network by repeatedly connecting and disconnecting clients. On the right, the Remote Laboratory Interface setup of the experiment is shown. Since both the Client Manager and the Network Manager connect to one router, this increases the amount of strain put on the network. Note that each of the machines hosting "clients" host 50 copies of the dummy client to help increase strain put on the network. A fake avatar client was created which would connect and disconnect from the Forest network repeatedly and very quickly. These avatars connect at random intervals to create a steadier stream of control packets. This fake avatar sends no data packets, as they don't place any strain on the control elements. While running 300 of these avatars on a Forest topology with two routers and only one of each control element (Network Manager, Client Manager, and Comtree Controller), the following chart describes the strain placed on the control elements. Session requests per second describes the number of times each avatar requests a new Forest session times the number of avatars (300).

Offered load (average session requests per second)	Client manager CPU Usage	Network Manager CPU Usage	Average bandwidth on Client Mgr Link	Average bandwidth on Net Mgr Link
600	5%	9%	2,000 pkts/sec	5,400 pkts/sec
900	5%	13%	2,485 pkts/sec	7,450 pkts/sec
1200	6%	16%	3,200 pkts/sec	10,000 pkts/sec
1800	8%	20%	4,700 pkts/sec	15,000 pkts/sec
2400	10%	28%	14,500 pkts/sec	36,000 pkts/sec

While the CPU loads are very appropriate, the packet rates for these experiments far exceed expected packet rates. However, these control components are still capable of handling at least 30,000 pkts/sec, which is certainly enough for a large number of clients in our implementation of Forest.

Monitor



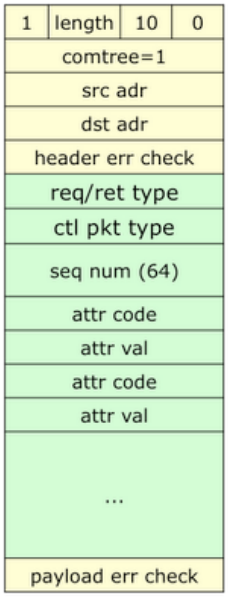
The Monitor is a client that provides data to the external *Show World Display*, shown below. Upon connecting to the Forest Network, it immediately subscribes to every multicast group within the comtree specified by the remote user, and sends information to the external display, which can then give a visual representation of the virtual world being simulated. The Show world display shows each grid square that is treated by the Forest network as a multicast group. On the right, there are 36 grid squares; hence there are 36 multicast groups per comtree. Each circle represents an avatar; the top left number is that avatar's forest address, the bottom left is the number of avatars currently being subscribed to, and the bottom right is the current number of avatars that can actually be seen by each avatar. Visibility here is based on walls in the virtual world. If one square can be seen from another, then all avatars in that square subscribe to the other square's multicast group. For example, let's examine avatars 3.2, 1.13, and 3.9. All squares in that particular region of the world subscribe to one another because there is a line of sight between them. Thus, each of those three avatars is subscribing to two others, but none can see each other due to walls.

Avatar

The Avatar is a specific client for a virtual world implementation of Forest. As with a real virtual world, the avatar is given a map to roam around in randomly. The map is given as a grid, with walls along specified edges of the grid. The avatar, upon starting up, calculates a mapping from square to square, defining which squares of the grid contain a location with line-of-sight to any other square. In addition, upon reaching one of these predefined walls, an avatar will bounce off of it and continue in the opposite direction. The avatar sends a *subscribe/unsubscribe* packet each time it reaches a new square. Currently, the avatar switches between data comtrees randomly throughout its run, in order to show the ease at which a client can switch between comtrees. The avatar also hosts a remote connection from the external *World Display*, allowing a user to connect and control the avatar with the arrow keys.

Control Packet Format

Control packets are separated into three types: in-band, client signaling, and network signaling. In-band control packets are used to forward packets in comtrees and by clients to connect to their respective routers. The format of the client signaling and network signaling is as shown on the right. Below is a description of the added fields.



Response Request Type (**RRT**). There are three allowed values here: *Request*, which specifies a request packet, *Positive Reply*, which indicates a positive reply to a previous request packet, and *Negative Reply*, which indicates a negative reply to a previous request.

Control Packet Type (**CPT**). This specifies the type of control packet. Control packet replies must have the same type as the request packet.

Sequence Number. The controllers use this field to indicate which request a reply is for. All control packets in a sequence should have the same sequence number.

Client signaling packets

Client Signaling packets, are identified by the packet type CLIENT_SIG (10). They can only be sent on comtrees numbered between 1 and 99.

- *Client Add Comtree* (10). Sent by a client node to a *Comtree Controller* to define a new comtree. Has no arguments. If successful, the reply packet includes the number of the allocated comtree.
- *Client Drop Comtree* (11). Sent by a client node to a *Comtree Controller* to remove a previously defined comtree and free all of its resources. Has a single required argument, the comtree number.
- *Client Get Comtree* (12). Sent by a client node to a *Comtree Controller* to retrieve various pieces of information about the comtree. Has a single required argument, the comtree number. The reply includes the comtree number, the comtree owner, the number of leaves and the internal and external bit/packet rates defined for the comtree.
- *Client Modify Comtree* (13). Sent by a client node to a *Comtree Controller* to modify various configuration parameters of a comtree. Has a single required argument, the comtree number. Optional arguments include the internal and external bit/packet rates.
- *Client Join Comtree* (14). Sent by a client node to a *Comtree Controller* to request addition to a specified comtree. Has a single required argument, the comtree number.
- *Client Leave Comtree* (15). Sent by a client node to a *Comtree Controller* to drop out of a specified comtree. Has a single required argument, the comtree number.
- *Client Resize Comtree* (16). Sent by the owner of a comtree to a Comtree Controller to request that the bit/packet rates of the comtree links be adjusted to reflect the access link rates of the leaves. Has a single required argument, the comtree number.
- *Client Get Leaf Rate* (17). Sent by a client node to a Comtree Controller to request the bit/packet rates assigned to a specified leaf node. Only the owner may request rates associated with other leaf nodes. Has two required arguments, the comtree number and the address of the leaf. The reply includes the comtree number, the leaf address and the rate parameters.
- *Client Modify Leaf Rate* (18). Sent by a client node to a Comtree Controller to change the bit/packet rates assigned to a specified leaf node. Only the owner may change rates associated with other leaf nodes and a leaf node may only reduce its own rates. Has two required arguments, the comtree number and the address of the leaf. The various upstream and downstream rate parameters are optional arguments.

Negative reply packets include a variable length error message terminated by a zero byte.

Network Signaling Packets

Network signaling packets are identified by the packet type *NET_SIG* (100). The payload of the packet carries the actual message content and has the same general format as the client signaling messages defined in the previous section. Network signaling messages pass only between network control elements and routers and are carried on comtrees numbered from 100 to 999.

There are several distinct groups of network signaling packets. The first is concerned with configuring router interfaces and is described below.

- *Add Interface* (30). Sent by a Network Manager to a router, to add an interface. Has four required arguments, the interface number, the IP address of the interface, the maximum bit rate for the interface and the maximum packet rate for the interface. A positive reply packet has no arguments.
- *Drop Interface* (31). Sent by a Network Manager to a router to remove a previously defined interface. Has a single required argument, the interface number.
- *Get Interface* (32). Sent by a Network Manager to a router to retrieve various pieces of information about an interface. Has a single required argument, the interface number. The reply includes the interface number, the IP address of the interface, the maximum bit rate and the maximum packet rate.
- *Modify Interface* (33). Sent by a Network Manager to a router to modify various configuration parameters of an interface. Has a single required argument, the interface number. Optional arguments include the maximum bit rate and the maximum packet rate.

The next group is concerned with configuring router links.

- *Add Link* (40). Sent by a Network Manager to a router, to add a link. Has five required arguments, the link number, the interface number that the link is being added to, the IP address of the peer, the Forest address of the peer and the type of the peer. There are no optional arguments. A positive reply packet has three arguments, the peer's Forest address, the link number, and the router's IP address.
- *Drop Link* (41). Sent by a Network Manager to a router to remove a previously defined link. Has a single required argument, the link number. A positive reply packet has no arguments.
- *Get Link* (42). Sent by a Network Manager to a router to retrieve various pieces of information about a link. Has a single required argument, the link number. The reply includes the link number, the interface number, the IP address of the peer, the port number of the peer, the type of the peer, the Forest address of the peer, the peer's allowed destination address, the bit rate of the link and the packet rate.
- *Modify Link* (43). Sent by a Network Manager to a router to modify various configuration parameters of a link. Has a single required argument, the link number. Optional arguments include the peer type, the peer port number, the peer's allowed destination address, the bit rate of the link and the packet rate. A positive reply packet has no arguments.

The next group is concerned with configuring comtrees at a router.

- *Add Comtree* (50). Sent by a Network Manager to a router, to add a comtree. Has one required argument, the comtree number. A positive reply packet has no arguments.

- *Drop Comtree* (51). Sent by a Network Manager to a router to remove a previously defined comtree. Has one required argument, the comtree number. A positive reply packet has no arguments.
- *Get Comtree* (52). Sent by a Network Manager to a router to retrieve various pieces of information about a comtree. Has a single required argument, the comtree number. The reply includes the comtree number, the value of the core Flag, the link number leading to the parent node in the comtree and the number of the queue used for the comtree.
- *Modify Comtree* (53). Sent by a Network Manager to a router to modify various configuration parameters of a comtree. Has a single required argument, the comtree number. Optional arguments include the value of the core Flag, the link number leading to the parent node in the comtree and the number of the queue used for the comtree. A positive reply packet has no arguments.
- *Add Comtree Link* (54). Sent by a Network Manager to a router, to add a link to a comtree. Has one required argument, the comtree number. Either the peer address or the link number is required as well. A positive reply packet has no arguments.
- *Drop Comtree Link* (55). Sent by a Network Manager to a router, to remove a link from a comtree. Has two required arguments, the comtree number and the link number. A positive reply packet has no arguments.
- *Resize Comtree Link* (56). Sent by a Network Manager to a router, to change the bit/packet rate of a link in a comtree. Has two required arguments, the comtree number and the link number. Optional arguments include the bit rates in the downstream and upstream directions (relative to the comtree root) and the packet rates in the downstream and upstream directions. A positive reply packet has no arguments.

The next group is concerned with configuring routes at a router.

- *Add Route* (70). Sent by a Network Manager to a router, to add a route. Has two required arguments, the comtree number and the destination address. Optional arguments include a link number and a queue number. A positive reply packet has no arguments.
- *Drop Route* (71). Sent by a Network Manager to a router to remove a previously defined route. Has two required arguments, the comtree number and the destination address. A positive reply packet has no arguments.
- *Get Route* (72). Sent by a Network Manager to a router to retrieve various pieces of information about a route. Has two required arguments, the comtree number and the destination address. The reply includes the comtree number, destination address, a link number and a queue number.
- *Modify Route* (73). Sent by a Network Manager to a router to modify various configuration parameters of a route. Has two required arguments, the comtree number and the destination address. Optional arguments include the a link number and a queue number. A positive reply packet has no arguments.
- *Add Route Link* (74). Sent by a Network Manager to a router, to add a link to a multicast route. Has three required arguments, the comtree number, the destination address and the link number. A positive reply packet has no arguments.
- *Drop Route Link* (75). Sent by a Network Manager to a router, to remove a link from a multicast route. Has three required arguments, the comtree number and the destination address and the link number. A positive reply packet has no arguments.
- *New Client* (100). Sent by a Client Manager to a Network Manager, to request information to add a client to the Forest network. Has two required arguments, the ip address and port number of the client. A positive reply packet has three arguments, the ip address of the router to connect to, the Forest address of that router, and the Forest address of the avatar.

- *Client Connect* (101). Sent by a router to a Network Manager, who forwards it to a Client Manager to ensure that clients are connecting to their routers. Has two required arguments, the Forest address of the connecting client and the Forest address of the router. A positive reply packet has only the Forest address of the router.
- *Client Disconnect* (102). Sent by a router to a Network Manager, who forwards it to a Client Manager to ensure that clients are disconnecting from their routers. Has two required arguments, the Forest address of the disconnecting client and the Forest address of the router. A positive reply packet has only the Forest address of the router.

The next group is concerned with booting routers

- Boot Request (120). Sent by a router to a Network Manager, requesting information from the topology to start up. Has no required arguments. A positive reply has the first leaf address and the last leaf address that the router is in charge of.
- Boot Complete (121). Sent by a router to a Network Manager, informing it that the router has completed booting from the information given in the boot request. Has no required arguments.
- Boot Abort (122). Sent by a Network Manager to a router that either cannot add its interface, or cannot add links as requested, telling it to quit booting and start over the process of booting. Has no required arguments

Future Work

We still have many goals for the Forest network before it is ready to be used in a commercial application. These include

- Creating a web interface for the client manager, through which a user can create a username and manage their account.
- Porting Forest to the Supercharged PlanetLab Platform. This is a nationwide set of computers on which we can run Forest to make sure it holds up well in a more lag-prone environment.
- Create a more realistic virtual world which users can log into more easily

The Forest network clearly supports a very large number of clients updating one another often and reliably. While we have focused on creating a virtual world implementation, another interesting implementation of Forest would be a sensor network, and it would be very interesting to see Forest used for such a purpose.