Terracast

Live Multicast on Wide Area IP Networks

Erik van Zijst

Fall 2004

Vrije Universiteit,

Faculty of Sciences, Department of Mathematics and Computer Science,

Amsterdam, The Netherlands

erik@marketxs.com

1. Abstract

Contents

1. Abstract....1
2.Introduction....4
3.Project Background....5
4. Problem Description....1
5.Terracast Architecture....7
5.1. High-Level Overview....7
5.2. Terracast Network Addresses....8
5.3. Protocol Endpoints....10
5.4. Router Packet Switching Core or Kernel....11
5.5. Interface Modules....12
5.6. Interceptor Pipelines....14
5.7.Client Programming Library....18
6. Routing on an Overlay Network....20
6.1. Overlay Network Topologies....20
6.1.1. Robustness....20
6.1.2. Dynamic Routing Algorithm....21
6.1.3. Shortest Path Routing....21
6.1.4. Efficient Multicast Distribution....22
6.1.5. Autonomous Overlay Topology Construction....24
6.1.6. Conclusion....24
6.2. Dynamic Routing....25
6.2.1. Link-State Routing....26
6.2.2. Distance-Vector Routing....26
6.2.3. Distance-Vector Versus Link-State....27
6.2.4. ExBF - The Extended Bellman-Ford Protocol....28
6.4. Scalable ExBF Routing in Large Networks....29
7. Single-Source Multicast Routing....40
5.4. Reliable Layered Multicast....50
5.4.1. Localized Packet Retransmission....52
5.4.2. Selective Packet Retransmission and Out-of-Band Reception....58
5.4.3. Restoring Global, Local, or No Packet Order....60

5.4.4. Layering Market Data.....64
5.5. Packet Prioritization and Multicast Congestion Control.....69
6.Related Work.....76
7.Evaluation.....77
8.Conclusions.....78
Bibliography.....79

2. Introduction

3. Project Background

The Terracast project was started almost two years ago as a research effort at MarketXS to see if it was possible to use the ever expanding, public Internet as an efficient, scalable backbone for distributing live market data across the globe. At the time MarketXS operated a centralized financial *ticker-plant* that was used by customers to obtain market data for use in their own backend systems or websites. A financial ticker-plant is a centralized system operated by a market data provider and used to process and distribute market data to clients. Data was accessed using a spcial API and exchanged over direct TCP connections. As the focus moved towards providing large volumes of live streaming market data to an increasing number of customers, MarketXS found itself distributing largely the same non-interactive content over the Internet to individual client applications using standard unicast TCP connections. The cost of both processing power at the distribution part of the ticker plant, as well as the necessary Internet bandwidth, increased linearly with the number of connected clients.

This is different to how larger competitors handle data distribution. Coming from a less globally wired past, the major competitors have traditionally used satellite broadcast systems to get non-interactive live content to their customers. Although this comes with fairly high initial costs, it scales practically infinitely with the number of clients.

MarketXS, being founded at the end of the Internet hype, has relied on the Internet as a flexible and cheap communications medium which appeared to be suitable for all but the most demanding institutions. However, as the amount of data and the number of clients grew, the need for a more scalable distribution mechanism emerged. Satellite was not favored, as that would invalidate the service's selling point that no client side hardware was required. Ideally, market data would be published to the Internet just once with a fixed cost, where it would be multicast to all paying customers. Unfortunately, since the Internet does not allow for such distribution natively, a virtual network was designed to reduce the amount of required bandwidth at the ticker plant without abandoning the Internet.

Because the Internet offers a best-effort delivery service of which the quality and throughput can greatly vary, the new virtual distribution network should be very robust with no single points of failure and should be able to constantly deliver its data with minimal delay, regardless of network congestion.

4. Problem Description

The Internet has been tremendously popular and has seen phenomenal growth during its few decades of existence. Especially since the birth of the World Wide Web the amount of online content has been spectacular and steadily growing. However, despite its versatility in digital communication, interoperability and internationally accepted communication protocols, its fundamental design has not changed much since its conception and it is easy to forget that the system does not excel in everything. Watching live TV, for example, is something which has still not happened over the Internet, even though television has been around almost twice as long as the Internet Protocol and represents a huge market. The technical reasons for this are fairly fundamental. This thesis will study the Internet's shortcomings regarding live content and in particular content that is to be delivered to thousands or more subscribers simultaneously. Also, having identified the problematic areas, it will introduce a software-based system that can run on top of the existing, unmodified Internet as a virtual network and alleviates some of the issues, making the network more suitable for live data distribution to large audiences.

The Internet is a packet-switched network where data is exchanged in small units or packets that are independently transported over the network. An important strength of packet-switching is that it allows for very flexible use of the physical network wires. When two communicating parties have no data to exchange for a certain period of time, no packets are sent and the wires can carry packets from other parties. More conventional systems such as the traditional telephone system, most cable television networks and radio are circuit-switched. They reserve a fixed amount of capacity or bandwidth for each communication session, regardless of the channel's use. This fundamental difference has important consequences for the type of applications they support best. The cable television network for example is perfect for delivering a live video stream of constant quality and without noticeable delay because the necessary bandwidth is reserved on the network and can never be used by other streams. This puts a hard limit on the total number of concurrent streams. On the Internet bandwidth is not reserved, but available to, and shared by, everyone. The consequence is that it cannot guarantee a minimum amount of end-to-end bandwidth, often causing live video streams to appear jerky and frames to be skipped.

Distributing real-time market data over the Internet to many concurrent receivers requires the network to consistently offer a substantial amount of bandwidth. Unfortunately the properties of the shared communications medium cannot guarantee such sustained performance and when the medium fails to meet the throughput requirements either temporarily or permanently, the service will suffer from delayed delivery or miss packets all together.

The most popular communication protocol over the Internet is the reliable TCP protocol. However, since this protocol only supports point-to-point connections, a unique connection needs to be setup from the source to each individual receiver. Not only does this waste bandwidth as each connection carries the same market data stream, the parallel unicast connections also have to compete for bandwidth, which can lead to permanent congestion and impact the timely delivery for all receivers.

The use of an Internet multicast protocol can offer substantial improvements over the use of parallel unicast TCP connections. Multicast subscription and routing protocols such as DVMRP [WAI88] and PIM [ADAM04], [EST98] allow for data packets to be sent to the network once while the network itself clones the packets as they are delivered to each subscribed receiver, making sure each participating network link carries only a single copy of each packet. Although this minimizes the bandwidth requirements, IP multicast, in contrast to TCP, does not offer guaranteed, nor in-order delivery. Hence, the use of IP multicast services does not necessarily make the Internet or any other wide-area, shared IP network suitable for distribution of real-time market data.

Following from these requirements are the features a packet-switched network will need to support in order to be a useable transport medium for dissemination of real-time, non-interactive content to large audiences.

Multicast Routing

At the very least, the network needs to support one-to-many communication in that it can send data packets from a data source to more than one receiver, ideally without putting extra stress on the network or source when the number of receivers increases. Multicast routing can be offered in various ways. The most common way is to let the receivers tell the network, but not necessarily the source, which streams they want to receive and let the network compute data

distribution paths that deliver the right packets to each receiver. Multicasting can also be done by letting the source encode the list of receivers in each data packet, thereby freeing the network from the potentially computationally intensive task of maintaining multicast distribution paths. Although this approach usually does not scale to large audiences, applications such as Mobile Ad Hoc Networks or MANETs [JI03] frequently use this and one approach has even been patented [MAR04]. A third approach places all logic at the receivers by letting the network apply a broadcast mechanism whereby each packet is delivered to every connected node and the receivers filter out only those packets that are interesting. Although this can be wasteful, its simplicity can still make it attractive especially in situations where bandwidth is abundant. Even though it does not use packet-switching, conventional radio and cable television do exactly this.

Adequate Flow Control and Timely Delivery

Because non interactive live data streams do not actively anticipate network congestion the way TCP does, the network will need to manage the available bandwidth in a way that allows for fair or equal division among the streams. Without this, high volume streams can consume a large capacity percentage of overloaded links and depending on the router scheduling policies, cause less active streams to suffer delayed delivery or packet loss. An alternative to letting the network components handle the flow control and congestion is to put the responsibility at the source and receivers. If this makes the streams well-behaved, network resources will be implicitly divided fairly, similar to when only TCP connections are used.

Shifting responsibility for flow control management from the network components to the communicating peers usually requires a form of feedback information from the network or the receivers. In this case it is important that the amount of feedback does not grow linearly with the size of the audience, as that would reduce the scalability of the multicast. Even when a scalable form of feedback information can be realised and the data stream adapts its transmission rate according to the network conditions, the problem remains that live streams often lose their value when they are slowed down and delivered late.

Guaranteed Delivery

It would be ideal if every receiver would receive the data stream without loss or corruption. However, when the content is live and cannot be slowed down, while the network has limited capacity, packet loss is difficult to avoid. Even when there is sufficient bandwidth at all times, store-and-forward packet-switched networks usually do not guarantee the delivery of all packets.

The main reason for packet loss is congestion, but additionally packets can get lost when a router goes down that has unprocessed packets in its buffers. In networks that use dynamic routing, packets can also be dropped when the routing forwarding rules change.

In cases where packets are accidentally lost, an end-to-end mechanism of retransmissions may be applied that can compensate for the loss, similar to retransmissions in TCP. However, since this requires a form of feedback information, it is again important for reasons of scalability that the overhead involved with retransmissions is not linearly related to the size of the audience. For TCP connections this is no issue, as they only support a single receiver.

Fortunately, end-to-end transmission feedback can be avoided in at least two ways. First, it is possible to let the network components keep a copy of the most recently forwarded packets and let them participate in retransmissions by intercepting the retransmission requests and servicing them locally. This approach has well-known applications in [SPEAK01] and [CHIU98], but may lead to aggressive storage and increased processing power requirements at the network components.

The second alternative to end-to-end retransmission requests is that of encoding redundant information in the data packets. If enough redundancy is encoded, a lost packet’s content can be entirely recovered from the extra information in the other packets. This technique is commonly know as Forward Error Correction and is applied in several systems and the subject of various RFC's including [LUB02] and [VIC02]. An interesting commercial application is offered by Digital Fountain whose FEC implementation encodes content in a way that allows clients to reconstruct the original data from any combination of received packets, equal in length to the original data [DF04]. The downside of FEC is that it comes with a constant level of bandwidth overhead that is related to the level of packet loss tolerance, regardless of whether packets are actually lost.

Whichever approach to packet loss is taken, all will fail in case of a live data stream that incessantly produces more bytes than the network can forward. When local or end-to-end

retransmission requests are used, the problem will even be exacerbated as the retransmission requests use extra bandwidth, causing more data packets to be lost, leading to even more retransmission requests.

Having identified the requirements for live multicasts over packet-switched networks, we can understand why the largest publicly available network, the Internet, is less suited for this. Although IP natively recognises multicast packets by their class-D destination IP address, actual multicast routing is disabled on a large part of the network, making it unavailable to almost all end users. As of 1992 an application-based overlay solution to IP multicast exists in the form of the MBone experiment [SAV96] that uses software based multicast routers to allow native multicast network segments to be connected to each other, even while the global carrier networks that physically connect those network segments do not support multicast. While the MBone is over 12 years old and still in use at the time of writing, its penetration is limited mainly to academic institutions and larger ISPs. Hence, native multicast routing is not widely available. However, even with a connection to the MBone, the issues of flow control and guaranteed delivery remain unsolved, seriously limiting the use of multicast traffic to a relatively small number of applications and content.

Given the requirements for live multicast and the lack of support for them by the Internet, it is concluded that the problem of large-scale multicast is better dealt with in the application layer, rather than on the network layer. This leads to the popular research area of application layer multicast where a virtual network is constructed from software based router nodes that employ their own routing algorithms and congestion control mechanisms to support efficient multicast distribution. Over the last decade or so, research in this area has lead to an impressive number of application layer multicast solutions. A modest and incomplete list of solutions includes Scattercast [CHA00], Narada [CHU00], NICE [BAN02], Bayeaux [ZHU01] and Overcast [JAN00]. The fact that new initiatives continue to emerge illustrates the difficulty of solving multicast on packet-switched networks in a generic way.

This text introduces another application layer multicast solution called *Terracast*. Although Terracast has many similarities with existing solutions, its specific focus on scalability and real-time data lead to a design that distinguishes itself from related work on a number of important points. As discussed in the previous chapter Terracast was designed to offer an end-to-end solution for efficiently multicasting live stock market data to potentially anyone connected to the Internet. The fact that live market data is in many ways more demanding and much less tolerant

to data loss than audio and video means that aside from multicast routing and flow control, Terracast must be capable of giving certain hard guarantees over what is delivered to receivers. If packets must be dropped due to congestion or other irrecoverable problems, it is crucial that this is done in a fully deterministic way that does not corrupt the financial data. Where a viewer of a film may accept the random loss of one or two video frames, this type of behaviour can wreak havoc in financial data when the missed packet contains the update of a rarely traded company.

The need for Terracast to be able to deterministically deliver only certain parts of a data stream when the network lacks sufficient capacity lead to the research area of layered multicast. With layered multicast, the packets of a live data stream are sent as individual streams. This allows receivers to subscribe to only those layers that the network can handle so that random packet loss can largely be avoided. For Terracast, an enhanced form of layered multicast is proposed that guarantees complete delivery for certain layers to avoid random loss altogether, making it suitable for market data.

Two core activities of Terracast can be isolated. The first is that of running and managing a robust and scalable overlay network that uses its own routing algorithms and supports multicast. The other core activity is that of managing flow control and congestion when live streams overload the network and ensuring layered multicast can be offered with guarantees.

Because Terracast combines two research subjects, it is difficult to find comparable, existing solutions. The evaluation of Terracast is therefore divided in two. The part that manages the overlay network topology and multicast routing protocol is compared to similar, existing application layer routing products, while the flow control, bandwidth allocation scheme and layered multicast protocol is compared to other existing layered multicast solutions from the recent literature.

5. Terracast Architecture

This chapter will describe the software architecture of the Terracast router daemons and is organized as follows.

Section 5.1. starts with a high-level overview of the software and introduces the different components. Section 5.2. explains addressing on the Terracast overlay network, while section 5.3. describes the various communication protocols a Terracast router daemon offers to its connected user applications. Among others, these include best-effort datagram unicast packets, best-effort multicast packet distribution and reliable, layered multicast distribution services. Section 5.4. will focus on the application router's core: its kernel. The kernel is the central part of each router and participates in all communication. Section 5.5. describes how the router's kernel is able to send and receive data packets from the overlay network through its interface modules, while section 5.6. offers a closer look at the core processing components in the interface modules: its packet interceptor pipelines. It will show how interceptors increase the flexibility of the overlay routers. Section 5.7. will explain how user applications can program against the Terracast router through a client-side library.

5.1. High-Level Overview

The Terracast network is an overlay network consisting of many software router daemons that maintain long-lived TCP connections among each other. This is depicted in illustration 1. This illustration shows a Terracast overlay network of four software router daemons, connected by four TCP connections. Which connections each router will engage in is configured at router startup, making the network's topology somewhat static as changing the overlay topology requires configuration changes to at least two router daemons.

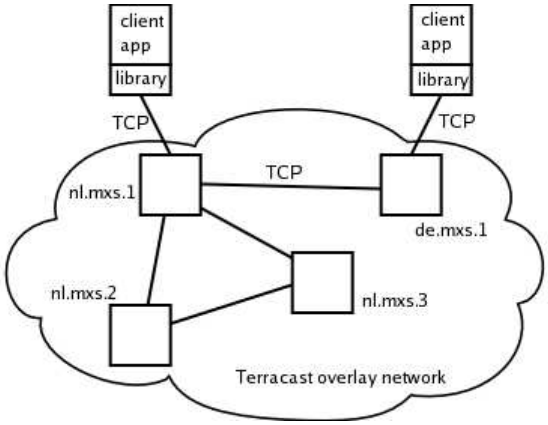


Illustration 1: A Terracast network is formed by software router daemons, interconnected by long-lived TCP connections.

The illustration also shows two connected user applications. User applications can use the overlay network for communication by programming against the Terracast client-side library which uses an RPC mechanism to relay the user's command and data packets to a single router instance. Router daemons and user applications have a one-to-many relationship where a single router can serve more than one user application at a time, but a user application can only be connected to one router.

5.2. Terracast Network Addresses

On Terracast, data packets can be addressed to a single communication endpoint, or to abstract multicast addresses. This section describes the role and representation of Terracast network addresses and introduces the concept of a *tport*.

Single communication endpoints are network addresses used by user applications. Access to them is exclusive, meaning that only one user application can use them for sending and receiving packets at a time. In that respect an analogy can be made between Terracast network addresses and those used on IP networks. An IP network address as used by user applications is the combination of a host IP address and a TCP or UDP port number. Together they form an address that is unique throughout the entire network[1]. They are suitable for sending packets to exactly one receiver.

Similar to IP, a unique network address on Terracast that can be used by a user application is the combination of the logical node address assigned to the overlay router and the port name chosen by the user application. If an application that is connected to a Terracast router with the logical node address "`nl.mxs.1`" wants to use port "`myport`" for receiving unicast data packets, the fully qualified network address becomes "`nl.mxs.1:myport`". Other applications connected to the Terracast overlay network that want to send packets to this application, will use this as destination address. To differentiate between ports in IP an Terracast networks, the remainder of this thesis will refer to Terracast port names as *tports*, rather than ports.

Just as with TCP ports, Terracast network addresses must be bound, prior to sending or receiving packets. When a data packet is sent from an endpoint address, it will contain this information as its source, allowing other routers or the receiving application to send a response.

1 Note that when IP masquerading or Network Address Translation (NAT) is used on IP, it becomes possible to share the same, masqueraded, network address at multiple locations.

Aside from network addresses that are bound and used by a single receiver application, Terracast allows packets to be sent to abstract multipoint destinations, or multicast addresses. A Terracast multicast address is a destination that everyone in the network can subscribe to. The network's software routers will make sure a copy of each data packet published to this multicast address is delivered to every user application that subscribed to it. Chapter XX will discuss the algorithms used to keep track of subscribed receivers and multicast routing. At this point it is sufficient to say that multipoint destinations are *single-sourced.* This means that only one user application can publish data packets to a multicast address, making Terracast suitable for one-to-many, but not for many-to-many communication.

The single source restriction is a result of the way multicast addresses are formed. Just as a unicast address, multicast addresses consist of both the node address of a router daemon and an available tport chosen by the user application that wants to publish. Hence, "`nl.mxs.1:mygroup`" could be a valid multicast address, used by the user application that bound the tport "`mygroup`" while being connected to router "`nl.mxs.1`". The node part of a multicast destination address is that of the router daemon used by source application, while the second part is that of the tport that the application chose.

Because only the tport section of a multicast address can be freely chosen (as long as it is not already in use by another application that is also connected to the same router daemon), each multicast address explicitly contains the source's location on the overlay network. While this makes multicast communication less flexible because publishing is not anonymous, it greatly simplifies subscription management. This will be covered extensively in chapter XX.

Both unicast and multicast addresses are syntactically equal. The address "`nl.mxs.1:timeservice`" could be a unicast addresses used and bound by an application that provides the local date and time in response to any data packet it receives. However, it could also be a multicast addresses that anyone can subscribe to, to receive periodic date and time broadcasts from the user application that bound this address as a multicast address. Because of this syntactic equivalence, each packet contains a flag that indicates whether its destination address should be interpreted as a unicast or multicast address.

5.3. Protocol Endpoints

The TCP/IP protocol suite comes with different protocol implementations that run on top of the packet-oriented IP base protocol. These are UDP and TCP. Their main difference is in the fact that TCP is connection oriented and reliable, while UDP offers a best-effort, datagram oriented service. Terracast takes a similar approach where several higher layer protocols are implemented as services over the packet oriented base layer. In Terracast these are referred to as protocol endpoints.

The current version of the platform comes with five standard protocol endpoints. Two offer unicast communication, while three provide various ways to do one-to-many data distribution. They are briefly summarized in table 1.

Protocol	***Description***
UUP	Unreliable Unicast Protocol. Offers best-effort unicast datagram services to user applications. This protocol is somewhat comparable in functionality to UDP.
RUP	Reliable Unicast Protocol. Offers reliable unicast communication between peers. The protocol is connection oriented and functionally comparable to TCP. A large part of its state diagram was taken from TCP.
UMP	Unreliable Multicast Protocol Offers best-effort multicast datagram services to user applications on the Terracast overlay network. It is the functional equivalent of UDP multicast packets on IP networks. It does not guarantee packet delivery, nor in-order reception.
OLMP	Ordered Layered Multicast Protocol Offers multicast communication with receiver-driven rate control. Complete delivery is not guaranteed, but the packets that are received are guaranteed to be in their original order. This protocol has no equivalent in IP and is connection oriented.

Protocol	*Description*
ROLMP	Reliable Ordered Layered Multicast Protocol Offers reliable multicast communication with receiver-driven rate control. Stream layering allows each subscriber to receive the data stream in the highest possible quality, while the source never has to slow down. This protocol has no equivalent in IP and is connection oriented. Both OLMP and ROLMP were designed for real-time data distribution.

Table 1: The Terracast overlay network offers five different communication protocols to its user applications. Two offer unicast, while three provide different types of multicast communication.

5.4. Router Packet Switching Core or Kernel

The heart of each Terracast software router daemon is the packet switching core, or kernel. With the exception of few specialized control packets, every packet that is received either from a static TCP connection to a neighbor router, or from a connected user application, passes through the switching core. Its task is to inspect the destination of each packet and use its routing tables to determine where to send it.

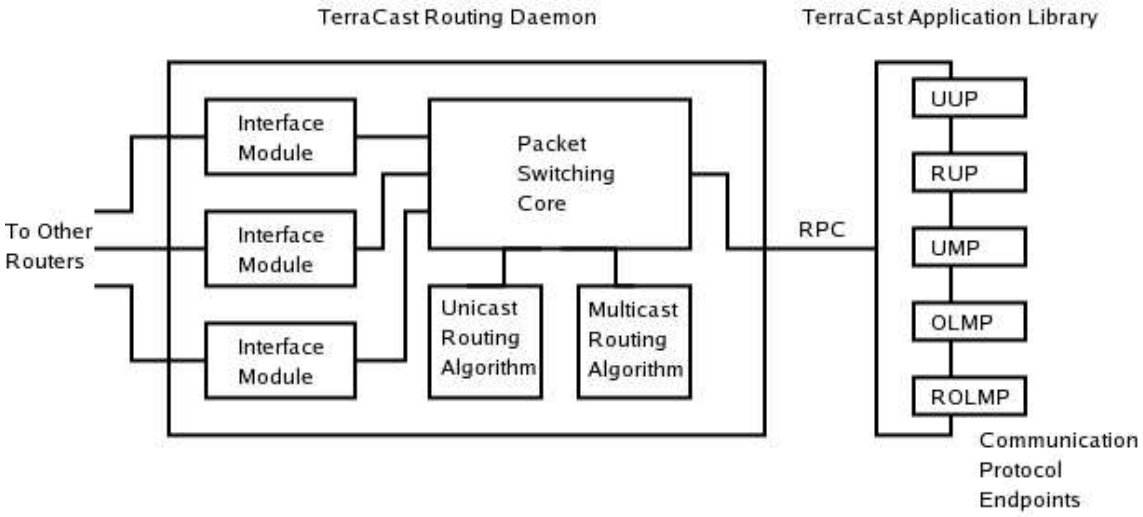


Illustration 2: Schematic view of a Terracast software router daemon. At the heart of the router is the packet switching core that is connected to the local user applications on one side and to the interface modules on the other.

The kernel's work runs in its own thread and is event driven. It remains idle until it gets notified either by an interface module that maintains the long-lived TCP connection with a neighbor router, or by a user application that is sending a packet. Because the kernel uses only a single thread, packet switching in Terracast is a serial process that will not benefit from multiprocessor hardware.

Illustration 2 shows where in the router daemon the switching core is located and which other components it communicates with. It clearly shows the central position it has and how it uses its unicast and multicast packet routing modules when processing packets.

5.5. Interface Modules

Each router daemon in the network is connected to one or more neighbors. This is done by establishing long-lived TCP connection between each other. A router daemon runs one interface module instance for each configured neighbor. The responsibility of an interface is to establish the TCP connection and to pass packets from the kernel to the TCP connection and vice versa. Packets of the first kind are referred to as outbound packets, while the latter are inbound packets.

For establishing the TCP connection to its neighbor, the interface module implements a simple algorithm. It first tries to connect to the configured neighbor router by actively trying to connect to its TCP/IP network address. If the connection is aborted with a connection refused or other error, it is assumed that the neighbor is not yet running, and the interface module starts listening on its configured IP port so that the neighbor can connect to it as soon as it is started up.
The interface module will only wait for an incoming connection request for a limited period of

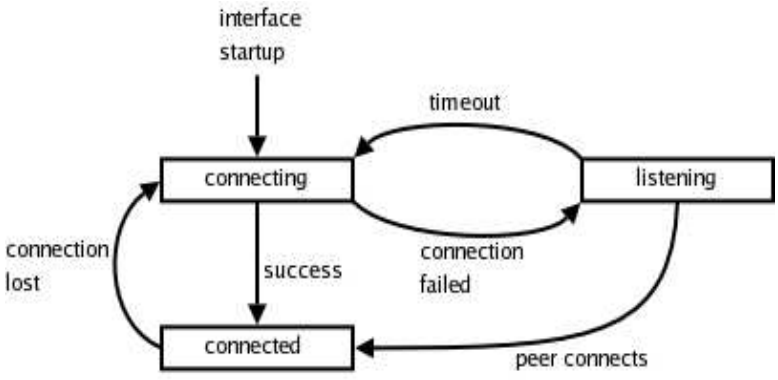


Illustration 3: State transition diagram for the interface module.

time. After that, it goes back into the state of actively connecting to the peer. To avoid the situation where both peers continue to switch states exactly at the same time, the duration of the listening state is influenced by a random factor. Although this does not guarantee that the algorithm terminates, it works well in practice. The interface state transition diagram is depicted in illustration 3.

The advantage of letting each peer switch between the roles of both client and server when establishing a connection, makes it possible to connect routers even when one of them is on a masqueraded IP network and is therefore unable to accept incoming TCP connections. Of course it is not possible to establish a connection between two routers that are both unable to accept incoming connections.

Interface module instances can be configured at router deployment through configuration files. This will make the router daemon automatically create the specified interfaces at startup. If the configured neighbor routers are online, all long-lived TCP connections will automatically be established. It is also possible to add new neighbor connections and interface module instances dynamically at runtime. This way the overlay network topology can be changed flexibly and new routers can be added to the network.

When an interface module manages to establish its connection with the neighbor router, they exchange their Terracast node addresses to inform each other of their presence. When the interface receives the node address of its peer, it passes this information, together with the announcement that the connection was established, on to the router's switching core. This information allows the kernel's routing algorithms to build dynamic routing tables. How routing is done inside the Terracast router daemons is discussed in chapter XX.

5.6. Interceptor Pipelines

The previous section explained that the role of the interface modules is to establish the connection with a configured neighbor router and to send data packets from the router switching core to the neighbor connection and vice versa. However, the interface module comes with a framework that allows custom software plug-ins to influence the stream of packets that flows between the network and the router's kernel. This mechanism is referred to as the interceptor pipeline. Each software plug-in component that needs to control the packet flow is a class that implements a simple interface. This interface allows interceptor instances to be chained together,

forming a packet processing pipeline. The contract of an interceptor is that it receives a packet, applies its operations and then passes the modified packet to the next interceptor in the chain.

The interceptor pipeline sits between the router switching core and the network connection. When the router kernel delivers a packet to the interface module for transmission to the neighbor router, the interface module runs the packet through the interceptor pipeline, giving the interceptors the chance to modify the packet. Each packet that comes out of the pipeline is transmitted to the neighbor router.

Each interface module has two interceptor pipelines. One is used to process outbound packets, while the other is used for inbound packets. These pipelines are independent of one another, meaning that not only the ordering of interceptors, but also the number of processing steps can be different for inbound and outbound packets. Also, because each interface module operates its own interceptor pipelines, they can be configured uniquely. This is illustrated in figure 4.

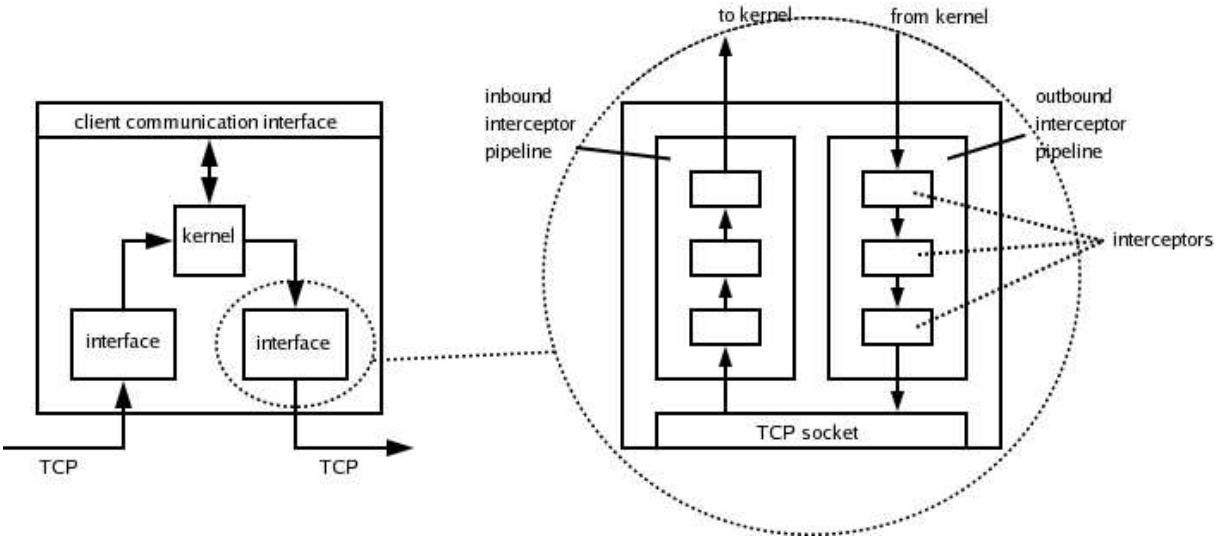


Illustration 4: A Terracast router daemon as shown on the left consists of a switching core or kernel and one or more interface modules that connect to neighbor routers. Using the unicast routing table, packets are forwarded from one interface to the other. The right hand side of the illustration shows the internals of an interface module. It shows how inbound and outbound packets are processed in separate interceptor pipelines and temporarily stored in packet buffers.

An example of an interceptor is one that filters packets coming from a specific Terracast router. When this interceptor is implemented as a manageable components that can be configured dynamically at runtime, it can be used to implement basic firewall functionality on the overlay network. When it receives a packet that matches its rejection pattern, it discards the packet by not passing it to the next interceptor in the pipeline. Another typical interceptor is a traffic

monitor that counts the size of each packet that passes by and uses this to log traffic activity and compute bandwidth statistics.

The use of interceptor pipelines in the interface modules is a powerful mechanism that was somewhat inspired by the concept of aspect oriented programming. AOP allows the functionality of objects to be extended dynamically [KIC97]. This is different from OOP that takes a static approach through inheritance. With this mechanism a Terracast router can be extended with additional functionality without modifications to the software.

As well as offering flexible extensibility, the interceptor pipelines have another important, secondary responsibility: they act as a packet buffer between the router kernel and the network. It was shown that the router's switching core runs its own thread and is notified by the interface modules when a new packet was received over the network. The kernel thread is woken up, reads the packet from the interface module and processes it. If the kernel decides that the packet must be sent out through another interface, it passes it to that interface and waits for the next notification. This sequence requires that passing a packet to an interface module is a non-blocking operation. However, writing data to the long-lived TCP connection can block when the TCP send-window is full. Therefore, the interface module must be able to temporarily buffer outbound packets to guarantee that the kernel thread is never blocked when it delivers a packet to the interface. Additionally, the interface module needs its own thread that continuously dequeues and serializes packets from the buffer and writes them to the TCP connection.

Thus, the secondary task of the interceptor pipeline is to provide that temporary packet buffer and offer the necessary inter-thread communication between the kernel thread and the interface thread that writes to the blocking TCP socket. To this end, interceptors are divided in two categories: those whose intercept method can block, and those that always return immediately.
The first category is referred to as blocking or synchronous interceptors. They use the calling thread to do their packet processing, as well as the delegation of the packet to the next interceptor. Control will not be returned to the caller until the packet was successfully delivered to the next interceptor. While the synchronous operation keeps the complexity of this interceptor low, the disadvantage is that it can keep the caller blocked.

Illustration 5a shows the use of blocking interceptors in the outbound packet pipeline of an interface module. It shows how the kernel passes a packet to outbound interceptor pipeline. The pipeline in this illustration contains only a single interceptor and it terminated by the “socket

interceptor". This is the pipeline's endpoint which does implements the interceptor programming interface, but does not delegate its packets to another interceptor. Instead, it writes them to the interface module's TCP socket. The illustration shows that the control is only returned to the router's kernel after the packet was entirely written to the TCP socket. How long a write operation on a TCP socket will block is unpredictable and has no upper limit. When the peer does not read bytes from the connection, the TCP send-window will fill up entirely, causing the operating system's kernel to block further write operations to the socket.

To avoid situations where the router's kernel is blocked for an arbitrarily long time, every interceptor pipeline should contain one non-blocking or asynchronous interceptor. A non-blocking interceptor guarantees immediate return of control to the caller by storing the packet in an internal buffer. These interceptors normally run their own thread to do the packet processing and delegation independent of the caller. This is depicted in illustration 5b.
A problem with blocking interceptors is that their packet buffer may grow indefinitely when the kernel delivers packets faster than the interface module's TCP connection can transmit. This is problematic as it requires infinite storage capacity. Also, it adds significant delay to the queued packets. Chapter XX will discuss several ways to handle situations where packets are delivered

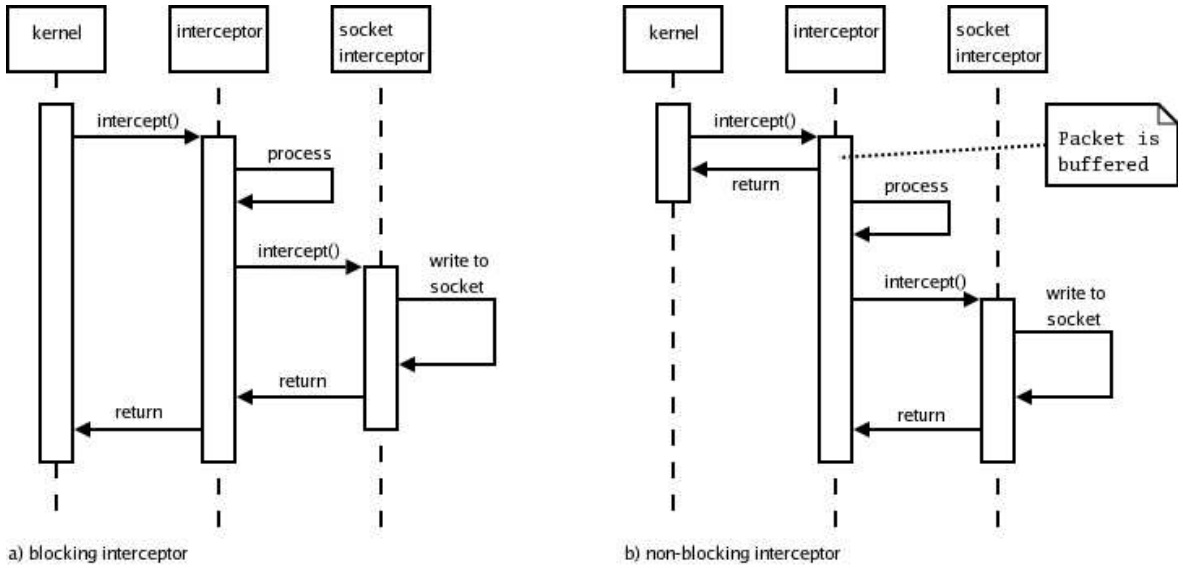


Illustration 5: Interceptor sequence diagrams. The left diagram shows the sequence diagram of a blocking interceptor. Because it uses the calling thread to do the packet processing as well as the delegation, the caller can be blocked. The diagram on the right shows how a non-blocking interceptor immediately returns control to the caller.

faster than they can be processed. At this point it is sufficient to say that each non-blocking buffer interceptor will discard packets when the size of its buffer exceeds a certain threshold.

Normally, an interceptor pipeline consists of zero or more blocking interceptors and one non-blocking interceptor. Because the non-blocking interceptor has a packet buffer that may discard packets, interceptors that measure actual traffic throughput would typically be situated after the non-blocking interceptor, while a firewall interceptor would usually sit in front of it.

5.7.Client Programming Library

The Terracast overlay network is accessible to user applications through the use of the client-side programming library. This library was already depicted in illustration 2 on page 11 and offers a local programming interface to applications.
The library connects to a router daemon at application startup and communicates with it using RPC. While the router daemon only provides basic functions for sending and receiving packets and for binding tports, the client-side library offers much richer functionality through the Terracast protocol endpoints that were introduced in section 5.3.

The RPC communication between the library and the router daemon is driven by the client. This means that only the client invokes functions in the router, but not vice versa. When the router's switching core receives packets addressed to a tport that is bound by the client application, it stores them until the client actively picks them up. Hence, receiving packets is realized by letting the client continuously poll the router for packets. To minimize the overhead of the RPC polling mechanism, the router's poll function does not return until there is at least one packet delivered by the switching core. When more than one packet is waiting, the poll function will return them all as soon as it is invoked.

The reason behind the polling nature of the client library is network related. An alternative would have been to let the client library expose a notification function for receiving packets. The client would then send a marshaled stub to the router daemon when it first connects. This stub can later be invoked by the router daemon as packets are received for the client. However, in most RPC implementations the service provider's object skeleton listens on a TCP port for incoming invocations. Because a Terracast router daemon will not always be running on a client's local machine, it can connect to any nearby router daemon that is willing to accept client connections. It is possible that this leads to a client application that runs on a masqueraded IP

network segment, but connects to a router daemon that is not on this segment. In this situation, the nature of IP masquerading prevents the router daemon from initiating TCP connections to the client. Giving the router daemon a stub for delivering packets will therefore not work. Because it is difficult for the client to detect the difference between this problem and the situation where the router simply has no packets to deliver, Terracast avoids it by letting the client take the initiative in all communication between router and client.

The router daemon uses a packet buffer to temporarily store packets for each connected client application until they are picked up. To realize this, the router uses an interceptor pipeline similar to those used in the interface modules. This is illustrated in figure 6.

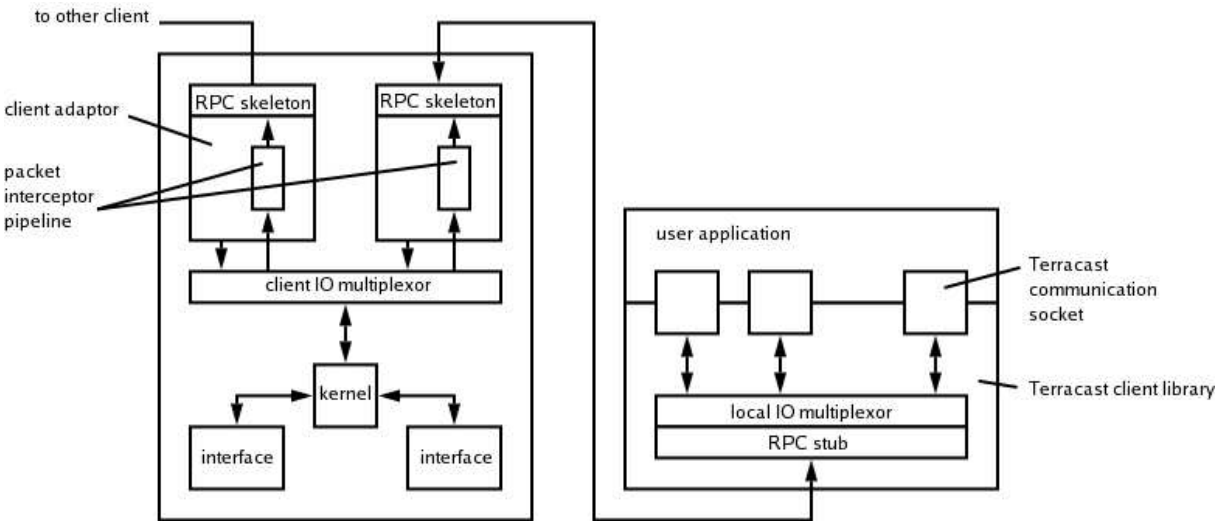


Illustration 6: User applications use the Terracast network through RPC using a client library. For each connected client, the router runs a client adaptor. When packets arrive addressed to a connected client, the router's client IO multiplexor passes it to the corresponding client adaptor where the packet is run through a packet interceptor pipeline and temporarily stored in a non-blocking buffer interceptor until the client library actively fetches the pending packets.

6. Routing on an Overlay Network

This chapter discusses some of the properties overlay network topologies. In section 6.1. it will be shown that the topology chosen for the network influences performance and resilience and how a proper topology is found. It will also show that the overlay network requires an adaptive shortest path routing algorithm to meet the Terracast requirements.
Section 6.2. will focus on this routing algorithm. It explains the required properties and compares these to existing routing algorithms. When an existing algorithm is selected, it will be extended with functionality that allows it to scale to large networks.

6.1. Overlay Network Topologies

The previous chapter explained that the Terracast router daemons form an overlay network. The topology of this network is defined by which router daemons are configured to be neighbors by sharing a TCP connection. Changing the neighbor configuration changes the overlay topology. This makes the overlay topology independent of the topology of the underlying IP network.

6.1.1. Robustness

Changing the topology of the overlay network by adding or removing links (edges) is easy and cheap, however these changes have impact on the network's performance and robustness. For example, an overlay network with a minimum number of edges, where the nodes are connected in a long chain, is vulnerable to node failure. If any node other than the chain's head or tail fail, the overlay network becomes partitioned. Hence, to make an overlay network robust, it should have enough redundant edges to survive node failure. On a network with redundant links, there is more than one path (sequence of edges) between two nodes.

When redundant links are required to make the overlay network robust and when adding links is cheap because it only involves configuration changes, an overlay network could be configured as a fully connected graph where each node is a direct neighbor of all other nodes.
While this may be a possibility for some networks, it is impractical on networks where new nodes frequently join and leave. Also, on large networks with thousands of nodes, maintaining a fully connected graph may be prohibitory expensive for the operating systems used on the

machines. Because Terracast must be able to support networks of more than thousands of nodes, a fully connected graph is not a suitable overlay topology.

6.1.2. Dynamic Routing Algorithm

In a robust overlay network that is not configured as a fully connected graph, the nodes do not have a direct TCP connection to every other node. Therefore they need routing information to decide how to reach nodes that are not directly connected to them. In simple overlay networks, this information could be a statically configured table that says which TCP connection should be used to reach each individual node in the network. In a robust network however, that is able to survive the crash of one of more nodes, the information in this routing table must be dynamic. When a TCP connection is lost because the neighbor crashes, the table must automatically find an alternative route for all destination addresses that became unreachable after the crash.

What is necessary in a large and robust overlay network is a routing algorithm that has enough knowledge of the overlay topology to select an alternative path when a destination becomes unreachable after the crash of a neighbor.
When it is important that data packets are routed through the network as quickly as possible, it would be beneficial if the routing algorithm is able to always find the TCP connection that offer the fastest path to the destination. Because many existing routing algorithms search for the fastest or shortest path to each destination and because Terracast targets the distribution of real-time data, Terracast uses a shortest path routing algorithm to maintain its routing tables.

6.1.3. Shortest Path Routing

In many algorithms, the definition of the shortest path between any two nodes is the path that contains the least number of edges. This technique is referred to as minimum hop routing. In networks where not all edges offer the same capacity and performance, minimum hop routing is inadequate. To cater for this, more sophisticated routing algorithms measure the actual propagation time, or end-to-end delay of each edge and use this to find the sequence of edges between any two nodes that offers the least total end-to-end delay.
The use of the latter technique, rather than minimum hop routing, is particularly useful on overlay networks where the capacity of the edges can greatly vary and even fluctuate over time. Because the TCP connections of an overlay network must compete for bandwidth with all other

networked applications, the actual performance of any overlay edge changes over time.
Because of this dynamic nature of overlay edges, Terracast uses a shortest path algorithm, based on minimum delay routing, that constantly measures the capacity of each edge and adjusts the routing tables accordingly.

6.1.4. Efficient Multicast Distribution

Although multicasting is discussed in detail in later chapters, it is briefly introduced here. The reason for this is that efficient multicasting on an overlay network puts some constraints on the overlay topology. To achieve optimal performance, the overlay topology should match the topology of the underlying IP network as closely as possible. This is illustrated in figure 7 which shows how a mismatch between physical (IP) and overlay topology results in poor overlay performance for multicast distribution.
The illustration shows five locations (buildings or departments) of a company network that are connected by five physical wide-area links. This physical topology is illustrated in figure 7c. An overlay network is now run on top of this company network and an overlay router daemon is deployed in each department. The router daemons are connected according to the overlay topology depicted in 7a.

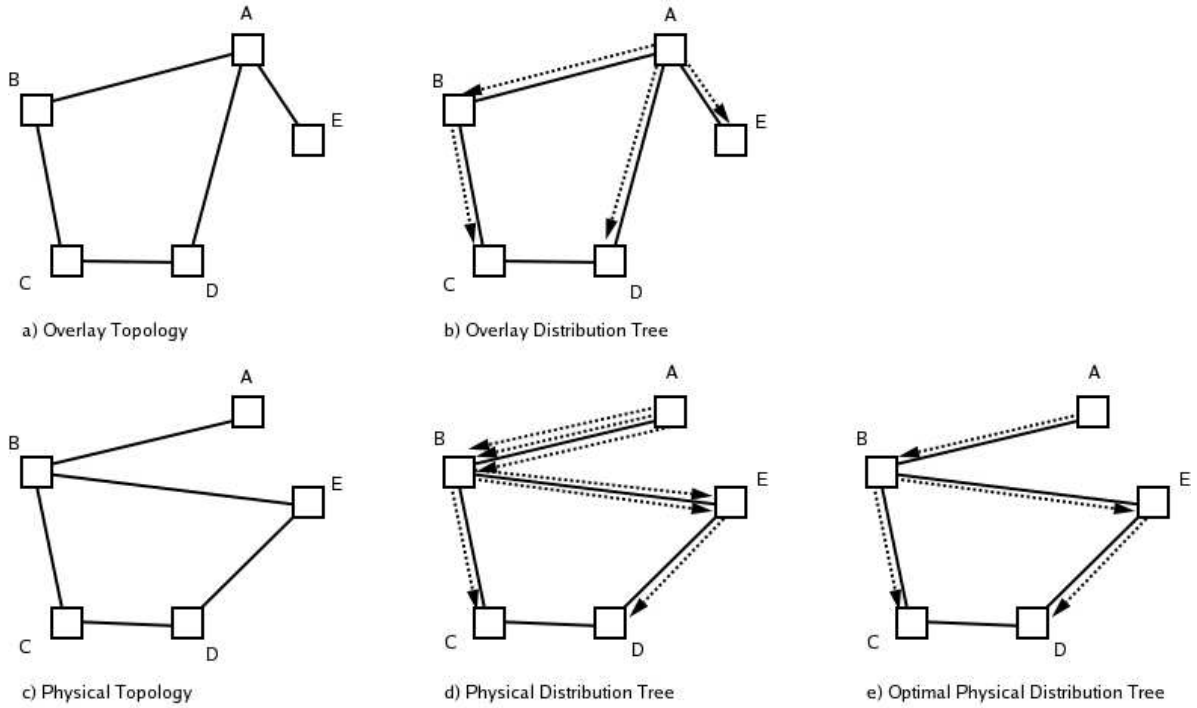


Illustration 7: Finding the best overlay network topology for an overlay network that is used for multicasting requires some knowledge of the underlying physical network.

With the physical network topology of figure 7c and the overlay topology of 7a on top of it, it is possible that the shortest path routing algorithm on the router daemon in department A measures that the other nodes in the overlay network are best reached through the paths that are represented by the dotted arrows of figure 7b. Hence, data packets addressed to nodes B, D and E are sent directly over A's TCP connections, while packets for node C are sent via B.

If at this point it is assumed that the multicast distribution tree is derived from the shortest path information in the routing tables (as many multicast routing algorithms do), it is likely that multicast traffic from A to all other nodes follows the dotted arrows of figure 7b. If all nodes are subscribed to the multicast session, A transmits three copies of each data packet: one packet to each of its neighbors. Node B sends a copy of each packet it receives to C.

Because the physical topology does not have a direct link between departments A and E, the TCP connection between the overlay router daemons at A and E runs via department B. Because department A only has a single physical link, all three copies of a multicast data packet will traverse this link. This is illustrated in 7d. This figure shows that the physical link between A and B carries three copies of the same multicast data packet and the physical link connecting B and E carries the same packet twice.

The redundant packets are an indirect result of the mismatch between the overlay topology and the topology of the underlying physical network and have a negative impact on the overall performance. In contrast, if the overlay network between the departments was configured to resemble the topology of the physical links, this would result in the multicast distribution tree that is depicted in illustration 7e. In this network, no physical link carries duplicate packets.

6.1.5. Autonomous Overlay Topology Construction

In the current Terracast implementation, configuration of the overlay topology is a manual process. To ease the administration of the network, the process of setting up and maintaining the topology could be implemented as an autonomous algorithm that is able to find the best router pairs. If this algorithm is run continuously in the background, it can even increase the resilience of the network by creating alternative edges after a the crash of a router daemon.
However, the overlay network runs on the application layer, independent of underlying network. Without creating a coupling between the underlying network and the overlay router daemons, the independent nature of the application layer makes it difficult to determine whether a TCP

connection to a certain router daemon will run over a unique physical path through the underlying network. Because of these difficulties, autonomous topology construction is not part of this thesis.

6.1.6. Conclusion

This section on overlay network topologies is concluded with a summarization of the properties an overlay topology should offer and what conditions must be taken into account when such topology is designed. These properties and conditions apply to a resilient overlay network that is targeted at the large-scale multicast distribution of real-time data. Removing and adding new edges to the overlay network is considered a manual process at this time. Also, the network must be able to scale to thousands of nodes and that propagation delay of packets should be minimized.

1. It was shown that in order to make an overlay topology resilient to the crash of one or more nodes, each node should be connected to more than two neighbors. Such a network offers more than one path (sequence of edges) between each pair of nodes.
2. The network must be able to scale to thousands of nodes and therefore the process of systematically connecting each node directly to all other nodes is not practical.
3. Because a node is not directly connected to all other nodes, it needs routing information to reach non-neighbor nodes.
4. As low propagation delay for packets is important in a network that is used for real-time streaming data, a routing algorithm should use performance measurements to determine the shortest path to any destination.
5. To allow for automatic re-routing of packets whose data path disappeared after the crash of a node, the routing algorithm should constantly evaluate and adjust the routing information. This is especially useful on overlay networks where edge performance fluctuates.
6. The overlay topology should closely match the topology of the underlying physical network to minimize the number of duplicate multicast packets on each physical link and to maximize multicast performance.

6.2. Dynamic Routing

The problem of finding the shortest path between any two nodes in a network of arbitrary topology without prior knowledge of the network's layout has been solved long ago.

When packet-switching was first introduced by Paul Baran and then independently a few years later by Donald Davies, its focus lay on military applications where it allowed networks with many redundant links and routers to survive destruction of a significantly large part of the network, without the need for manual intervention and reconfiguration. This required routing algorithms running at the nodes and through ARPA, Baran's early work [BAR64] influenced today's protocols.
Currently two major classes of dynamic routing protocols exist that differ in the way shortest paths are computed and what routing updates are propagated to other routers. The first class of protocols is known as *distance-vector* protocols, the latter as *link-state* protocols.

This section discusses the merits and liabilities of both types of routing protocols. It then selects an existing protocol that best suits the Terracast requirements.

6.2.1. Link-State Routing

Link-state protocols take the relatively simple approach of propagating the state of the entire network as a list of all known links in the network with their cost value. This way, every node in the network will have complete knowledge of the topology of the entire network. Changes to the topology can be propagated quickly to all nodes as the routing updates can be forwarded unchanged. Finding the next hop in the shortest path between any two nodes can then later be done by running Dijkstra's algorithm [COR90] on the network topology information.
To store the entire network topology in memory, the typical space complexity of a link-state protocol is $O(N^2)$ where N represents the number of nodes in the network. Successful link-state protocols include OSPF (Open Shortest Path First) [MOY89] and IS-IS (Intermediate System to Intermediate System) [IEEE90].

6.2.2. Distance-Vector Routing

Distance-vector protocols are based on the Bellman-Ford protocol [FORD62]. They differ from link-state protocols in the kind of information they exchange with their neighbors, as well as the amount of information stored in routing tables.

The original Bellman-Ford protocol tells its neighbors how far away, in terms of intermediate hops, it is from all the other known nodes. Effectively it publishes the length of every shortest path it knows. This is done as a collection of tuples, where each tuple contains a destination address and a path length. These tuples are called *distance-vectors*.
The receiving node increments the length of every path by one and compares each path to its own shortest path information. When it turns out the neighbor offers a shorter path to a certain destination, it replaces its own path with the new path. From now on, data packets addressed to this destination will be routed via this neighbor. Every time the information in a node's routing table is changed, it will publish its shortest paths to its neighbors. The presence of new nodes is automatically learned when a set of distance vectors contains a new destination address. Some widely accepted implementations of distance-vector protocols include RIP [HED88], [MAL98] and Merlin-Segall [MS79].

An important consequence of the way routing information is exchanged is that the information is first processed and merged into the local routing table, at which point a new routing update is derived from it. Because of the processing time required at each node, propagation of routing changes through the network is slower than with a link-state protocol.
An advantage of distance-vector protocols is that they maintain relatively little state information. The space complexity for this class of protocols is typically $O(N * e)$ where e is the average number of neighbors per node, while N represents the total number of nodes in the network.

Although the use of localized information and routing updates makes the protocol friendly in terms of storage requirements, it is susceptible to long-lived routing loops. This is probably the biggest disadvantage of the classical Bellman-Ford protocol. The loops occur when a route “backwashes” from a receiver to a sender after a link or node failure. This continuous exchange of an invalid route leads to an explosion of routing updates whereby the length of the invalid path is constantly increased. Because of its nature, this problem is known as *counting-to-infinity*. The effects of this problem can be reduced by putting an upper limit on any path length. This approach is taken by RIP, which does not allow paths longer than 15 hops. Other well-known

techniques include *poisoned-reverse* and *split-horizon*. Both prevent the "backwashing" of routes. The first explicitly advertises a destination as unreachable to the neighbor that provided the best path to the destination, while the latter omits this destination in its routing update to the neighbor. The counting-to-infinity problem and its solutions are further described by Bertsekas and others in [BERT87].

6.2.3. Distance-Vector Versus Link-State

Whether a link-state protocol is better than a distance-vector depends on the characteristics of the network. It is interesting to see that the Internet's predecessor, the ARPANET, initially used the Bellman-Ford-based RIP protocol, but was replaced by the SPF (Shortest Path First) link-state protocol in 1979 [MCQ79] because of its slow convergence after almost a decade of research. In 1987, the protocol was given a revision that made its reaction to topology changes less aggressive by applying exponential averaging over link costs [KAHN89]. After SPF, OSPF was introduced and gone through several versions. In 1998, OSPF Version 2 [MOY98] was introduced and remains one of the most widely deployed protocols for intra-domain routing. In 1999 OSPF Version 3 [MOY99] also provided support for IPv6.

After RIP was replaced by SPF in 1979, many new distance-vector protocols have been designed that structurally solve the long-lived loops of Bellman-Ford. The Merlin-Segall protocol that was mentioned previously is one of them. Another important one is the Extended Bellman-Ford or ExBF protocol proposed by Cheng and others [CHE89]. Merlin-Segall avoids all routing loops, both short-lived and long-lived, while ExBF only prevents all long-lived loops.

A comparison by Shankar and others [SHAN92] between the SPF link-state protocol and the newer, loop-free distance-vectors Merlin-Segall and ExBF showed that ExBF performs as good as SPF under most circumstances with a much lower space complexity. Merlin-Segall generally performs worst. Following the outcomes of Shankar's comparisons between link-state protocols and the more modern, loop-free distance-vector protocols, Terracast adopts the Extended Bellman-Ford protocol as the basis for its overlay routing.

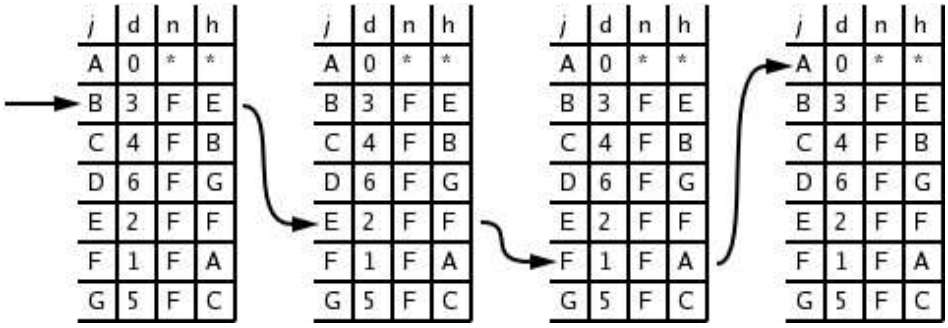
6.2.4. ExBF - The Extended Bellman-Ford Protocol

Because the ExBF distance-vector protocol is used as the base routing layer on the Terracast overlay network and because it will be extended with an address summarization mechanism to

improve performance on large networks, it is important to understand how the protocol manages its routing information. The protocol is therefore briefly discussed in this section.

ExBF distinguishes itself from the traditional Bellman-Ford protocol by storing not only the length and destination of each path in its distance table, but also the pre-final node of every path. The pre-final node of a path is the last intermediate hop a packet visits before it reaches its destination node. Using the pre-final address of a path, a router can backtrack the entire path to any destination by recursively looking at the pre-final node of a destination and treating it as a new destination. When an ExBF router exchanges distance-vectors with a neighbor, these vectors also contain the address of the pre-final node of every path. This makes a distance-vector in ExBF a triple, rather than a tuple.

The process of recursively tracing an entire path is illustrated in figure 8 where the routing table of node A is given. Tracing the full path from this node to destination B goes by first looking up the pre-final node on the path to B which is E and then using E as a new destination. The pre-final node of the path to E is F, while the pre-final node to F is node A itself. This concludes the reverse trace of the path that leads to A, F, E, B.



To avoid long-lived routing loops, ExBF uses its pre-final node information to apply the following trick: when node *i* advertises a routing entry to destination *j*, it refrains from sending it to any neighbors whose address is in the path n_0, n_1, ..., n_r, *j*, where n_0 is *i*'s next hop to *j* and n_r is the pre-final node of the path. For the example of illustration 8 that means that node A will not advertise its path to destination B, E or F to its neighbor F. Instead, it will explicitly report that A cannot reach those destinations.