Software Architecture of a Turtle Node

Petr Matejka, pmatejka@ontheweb.cz
7th March, 2004

1. Introduction

This document gives brief overview of software architecture of a Turtle node. Turtle nodes operate in peer-to-peer network described in [1], however several modifications of communication protocol were accepted. Most important changes are described in sections 1.1 and 1.2.

1.1. Anonymous addresses

First change is related to node addressing. In original protocol all nodes of query broadcast tree must remember part of routing information(immediate neighbour) for the query. Routing information is used in data retrieval phase to route retrieve request and response. In modified protocol the routing information is not spread in the network, but it is propagated back to query initiator. Each network node then remembers only routing information of its own queries. Although the information is bigger, nodes can easily decide what to remember and how long to remember it, because they know what they need to do with query responses.
Following example illustrates how the modified protocol works:
Node 0 initiates a query, which is broadcasted to many nodes, e.g. node 1, node 2 and node 3, see Figure 1. Node 3 finds a hit and informs node 2 about the hit (by response packet). Node 2 adds routing information – link number encrypted by node's secret key E(*key2*, *link1*) – and forwards extended response packet to node 1. Node 1 extends routing information to E(*key1*, *link3*):E(*key2*, *link1*) and forwards the response packet. Node 0 extends routing information to final form E(*key0*, *link3*):E(*key1*, *link3*):E(*key2*, *link1*), which is returned to query initiator along with the hit description (attributes etc. as described in Turtle paper).
Returned routing information allows node 0 to contact node 3, although nodes 1, 2 and 3 don't store any state. Because each node understands only its own part of routing information, it is anonymous and nodes know only their immediate neighbours on the path. We call this routing information **anonymous address**. Number of hops between nodes might be masked by other techniques.

1.2. Virtual circuits

All communication between Turtle nodes will use virtual circuits. It allows data of arbitrary length to be transferred over the network. When a node wants to establish a virtual circuit, it first needs address of target node. In our case the address is acquired by query protocol as described in section 1.1. Target address is then propagated along with connect request. Each node on the path unpacks (decrypts) its part of the address, so the circuit is established step-by-step to target node. Neither source nor target node know to whom they are connected. The data sent to the circuit by any of them is propagated step-by-step over the network.
Virtual circuits are also used for sending queries and query hits. They can be theoretically used for any type of application (not only querying and retrieving files) just like TCP/IP.

2. Components

In this document only communication infrastructure is described. GUI part of a Turtle node is left out. All main components of communication infrastructure are depicted in Figure 2: Router, TCP Channel Manager, Service Address Resolver, Query Service, Download Server and Client.

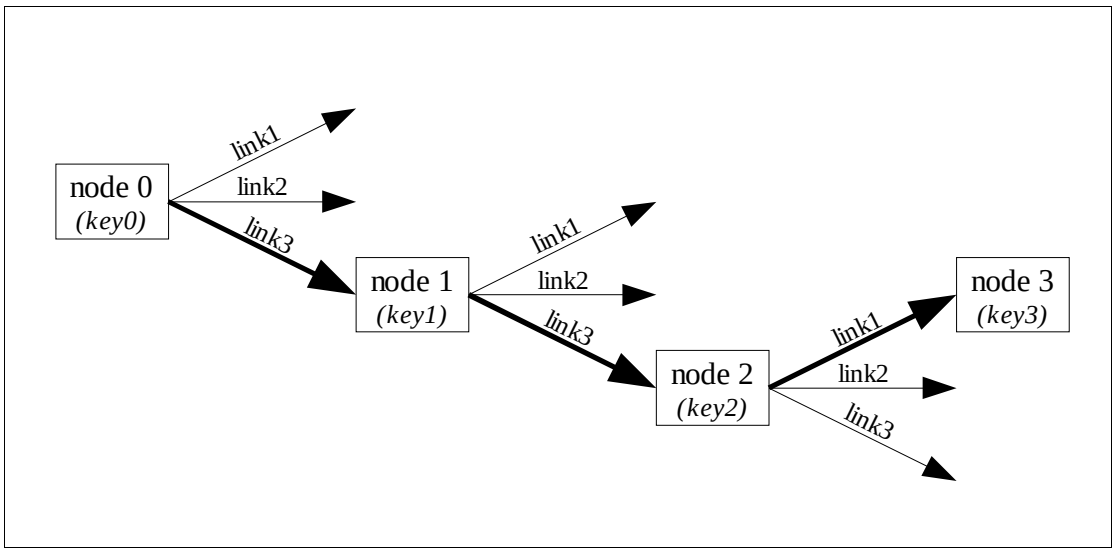

Figure 1 Query propagation

These components are connected by `Channel` interface. `Channel` interface includes methods for controlling virtual circuits:

- establish virtual circuit
- send data through virtual circuit
- flow control of virtual circuit
- close virtual circuit

There is also a method for discoverring services running on neighbour node. More detailes about this topic are given in section 2.3.

2.1. Router

Router is the central component of Turtle node. All data going through the node go through the Router. All other components are registered at Router and Router does circuit switching between them. We call these components **channels**. When virtual circuit connect request comes from one channel, Router decodes(decrypts) routing information and passes connect request further to another channel. When the circuit is established, Router forwards data between these two **circuit segments** until the circuit is closed. There can be many virtual circuits coming from/to one channel.
Each channel represents either local application or neighbour node. Router does not distinguish between these two types of channels. It simply forwards calls from one channel to another. Besides this Router provides the following services:

- register/deregister channel
- get channel list and their SAR addresses (discussed later)
- extend anonymous address by local routing information (needed to construct anonymous addresses e.g. by query service)

Internally Router creates one Router Channel object per registered channel. Router Channel provides `Channel` interface to registered channel and requires same interface from the channel. All calls to Router Channel are converted to command object and stored into command queue, which is serviced by Router thread. Router thread reads commands from command queue, analyzes them and forwards them as calls to `Channel` interface of registered channels.

2.2. TCP Channel Manager

TCP Channel Manager is responsible for communication with neighbour nodes. It listens for incoming TCP connections from neighbours and also periodically tries to connect them. If TCP connection is established and authenticated, Communication Channel object is created and registered at Router. From that moment it is possible to create virtual circuits through newly created channel.

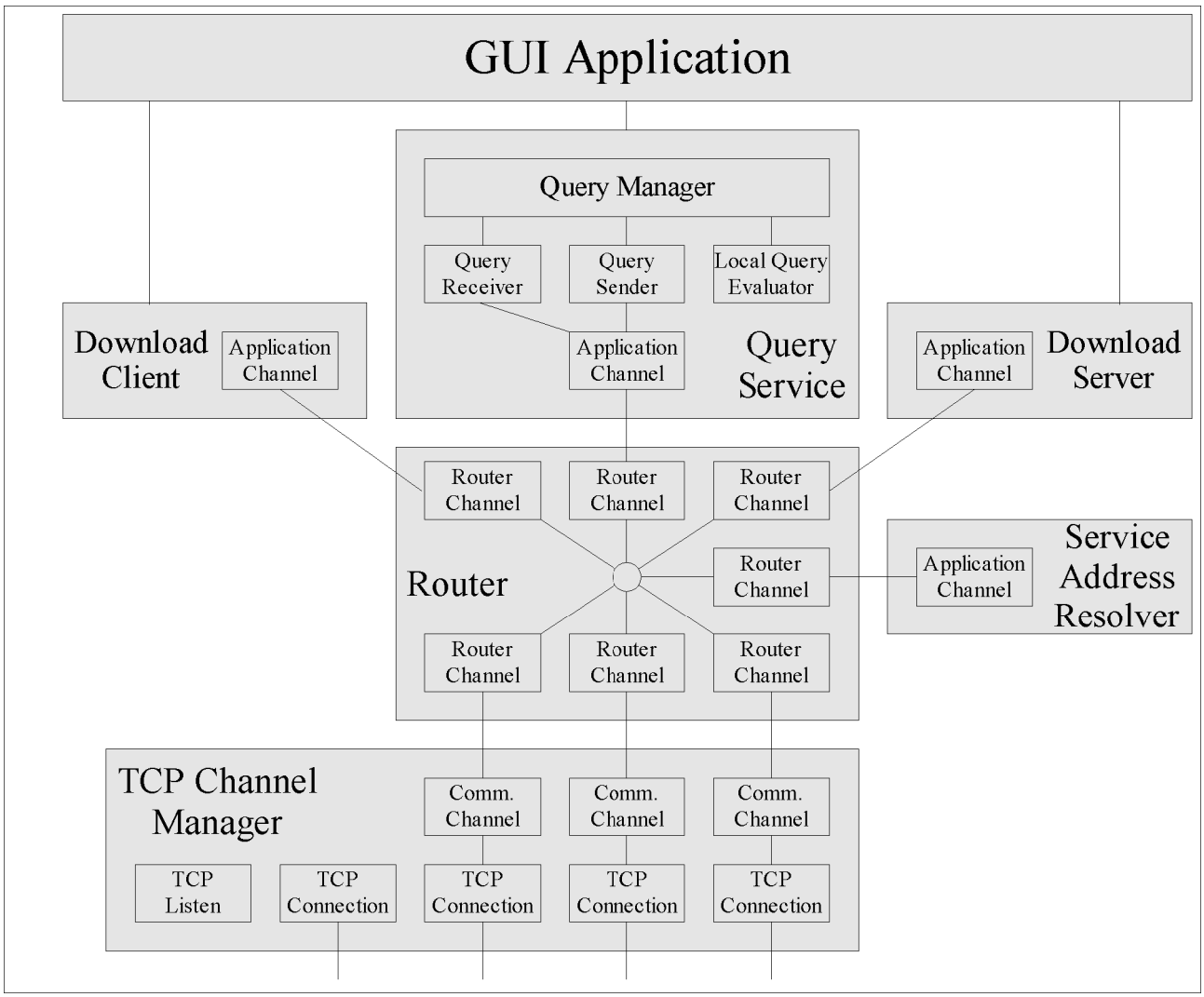

Figure 2 Components of Turtle node

All TCP communication is done from one thread. It receives data and passes them to Communication Channels. Communication Channel builds command from the data and passes it to Router (using `Channel` interface). On the other hand when Communication Channel receives call from Router, it converts the call to data structure and passes it to communication thread as raw data. Communication thread is then responsible for encrypting it and sending to neighbour node.
Communication Channel is also responsible for controlling the flow of virtual circuits. It supplies each circuit with send buffer. Using flow control call of `Channel` interface it does not allow sender to send more data than what can fit into the send buffer.
Because there can be many virtual circuits on one channel, data multiplexing is needed to guarantee fair division of TCP connection bandwidth. This is again up to Communication Channel. It cyclically sends data of circuits and assigns higher priority only to commands(connect, flow control, disconnect, etc.).

2.3. Service Address Resolver

So far no mechanism for discovering services running on neighbour nodes was introduced. It is neccessary to provide such mechanism, so that neighbour nodes can establish virtual circuits for basic services (currently only query service).
Each node has one special component called Service Address Resolver(SAR) that is registered at Router as channel with special flag. Router knows that it is SAR and provides its address through `Channel` interface to all other channels. Channels propagate the address to neighbour nodes. Routers of neighbour nodes store SAR address of each registered neighbour and provide it to local components.

SAR acts as register of local basic services. They can register at SAR with name and address. Same service running on neighbour node can ask its router for address of our SAR. Then it contacts our SAR and asks (by name) it for address of basic service. After extending acquired address by local routing information neighbour service can use the address to contact our service.

2.4. Query Service

Query Service component implements query mechanism described in [1]. Query Receiver subcomponent waits for incoming circuits from neighbour nodes. It is registered at SAR, so that neighbour nodes can find it. It receives queries on established circuits and passes them to Query Manager. When query hit is received from Query Manager, Query Receiver sends hit packet to circuit from which corresponding query came.
Query Manager receives queries from Query Receiver and from GUI. It keeps hash table of recent queries to be able to detect cycles in query graph and to know from where each query came. Queries not found in hash table are stored to it and are passed to Local Query Evaluator and Query Sender. Query hits received from Local Query Evaluator and Query Sender are passed back to Query Receiver or GUI.
Query Sender uses SAR to find Query Receivers of neighbour nodes and connects to them. Connections are used to send queries to Query Receivers and to receive query hits from them. When a query hit is received from neighbour node, anonymous address contained in hit packet is extended and then the packet is passed to Query Manager.
Local Query Evaluator evaluates queries on local filesystem and passes hits to Query Manager. Address contained in hit packet is set to address of local Download Server.

2.5. Download Server

Download Server is simple component that has only one purpose – to send requested file to virtual circuit. When contacted it waits for request consisting of file name and possibly also file position (to support continuing of broken downloads). It finds requested file on local filesystem and starts sending until the file is completely sent.

2.6. Download Client

Download Client communicates with Download Server running on remote node. GUI can ask Download Client to download a file. It must provide address of Download Server and name of the file, both can be obtained by querying. Download Client then contacts Download Server and sends request to download the file. When the file is received and stored to local filesystem, GUI is notified.

2.7. Other Components

There are a few other components not mentioned yet. They are subcomponents of some above-mentioned component.

2.7.1. Application Channel

Application Channel is used by local application components (Download Server/Client, Query Service, SAR) to bind to router using `Channel` interface. It provides BSD sockets-like interface to application components, so they can use virtual circuits very easily using well-known functions (connect, listen, accept, read, write, close, select).
Application Channel supplies each virtual circuit with receive buffer and optionally also with send buffer. Using these buffers Application Channel controls flow on virtual circuits.

2.7.2. Key Repository

Key Repository stores encryption keys and provides them to TCP Channel Manager. TCP Channel Manager uses them as master keys in TCP communication with neighbour nodes. Turtle relies on symmetric cryptography, so keys must be exchanged in advance and put into Key Repository before nodes are able to communicate.

2.7.3. Addressing Utility

This is a helper component for Router that can manipulate with anonymous addresses by the following operations:

- add record(piece of routing information)
- read record
- remove record

Different addressing schemes can be used in Turtle network by extending this component, e.g. addressing scheme that masks hop count.

2.7.4. Query Parser

When user enters a query, it is first parsed to binary form by Query Parser. Query in binary form is packed into query packet. Packet is passed to Query Manager that distributes it to the network. Distributing queries in parsed form saves CPU time of nodes, because parsing is done only once and not on each node.

3. Initialization and Configuration

Initialization procedure of Turtle node has these steps:

1. Router is created.
2. SAR is created and registered at Router.
3. TCP Channel Manager is created and registered at Router.
4. Neighbour nodes are one by one registered at TCP Channel Manager. TCP Channel Manager tries to connect them immediately and create Communication Channels for successfully connected neighbours. Communication Channels are registered at Router.
5. Download Server is created, its local directory is set and it is registered at Router.
6. Query Service is created, its local directory is set to same path as Download Server's path and it is registered at Router. Query Receiver registers itself at SAR. Query Sender tries to connect neighbour Query Receivers immediately.
7. Download Client is created and registered at Router.

From this moment, GUI can send queries to Query Manager and expect query hits from it. Query hits can be then browsed by user and selected files are downloaded using Download Client. Download Client notifies GUI about events as broken download, finished download or download progress.
GUI is responsible for storing the configuration in its configuration files (or elsewhere). It passes the configuration to communication infrastructure components via their interfaces during initialization or when user changes the configuration.

References

[1] Bogdan C. Popescu, Bruno Crispo, Andrew S. Tanenbaum: Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System