The internet Gopher protocol
a distributed document search and retrieval protocol

Bob Alberti, Farhad Anklesaria, Paul Lindner, Mark McCahill, Daniel Torrey
University of Minnesota Microcomputer and Workstation Networks Center
Spring 1991; Revised Spring 1992

gopher n. 1. Any of various short tailed, burrowing mammals of the family Geomyidae, of North America. 2. (Amer. colloq.) Native or inhabitant of Minnesota: the Gopher State. 3. (Amer. colloq.) One who runs errands, does odd-jobs, fetches or delivers documents for office staff. 4. (computer tech.) software following a simple protocol for burrowing through a TCP/IP internet.

Abstract

The internet Gopher protocol is designed for distributed document search and retrieval. This document describes the protocol, lists some of the implementations currently available, and has an overview of how to implement new client and server applications.

The protocol and software follows a client-server model. Documents reside on many autonomous servers on the Internet. Users run client software on their desktop systems, connecting to a server and sending the server a selector (a line of text, which may be empty) via a TCP connection at a well-known port. The server responds with a block of text terminated by a period on a line by itself and closes the connection. No state is retained by the server.

While documents (and services) reside on many servers, Gopher client software presents users with a hierarchy of items and directories much like a file system. The Gopher interface is designed to resemble a file system since a file system is a good model for organizing documents and services; the user sees what amounts to one big networked information system containing document items, directory items, and full-text searching capabilities across subsets of the information base.

Servers return either directory lists or documents. Each item in a directory is identified by a type (the kind of object the item is), user-visible name (used to browse and select from menu listings), an opaque selector string (typically containing a pathname used by the destination host to locate the desired object), a host name (which host to contact to obtain this item), and an IP port number (the port at which the server process listens for connections.) The user only sees the user-visible name. The client software can locate and retrieve any item by the trio of selector, hostname, and port.

In submitting a query to a search server, the client sends the selector string and the list of words to be matched. The response yields “virtual directory listings” that contain files matching the search criteria.

Distribution of this document is unlimited. Please send comments to the Gopher development team: <gopher@boombox.micro.umn.edu>. Experimentation with the mechanisms described here is encouraged.

1. Introduction

The Internet Gopher protocol is designed primarily to act as a distributed document delivery system. While documents (and services) reside on many servers, Gopher client software presents users with a hierarchy of items and directories much like a file system. In fact, the Gopher interface is designed to resemble a file system since a file system is a good model for locating documents and services. Why model a campus-wide information system after a file system? Several reasons:

(a) A hierarchical arrangement of information is familiar to many users. Hierarchical directories containing items (such as documents, servers, and subdirectories) are widely used in electronic bulletin boards and other campus-wide information systems. People who access a campus-wide information server will expect some sort of hierarchical organization to the information presented.

(b) A file-system style hierarchy can be expressed in a simple syntax. The syntax used for the internet Gopher protocol is easily understandable, and was designed to make debugging servers and clients easy. You can use Telnet to simulate an internet Gopher client's requests and observe the responses from a server. Special purpose software tools are not required. By keeping the syntax of the pseudo-file system client/server protocol simple, we can also achieve better performance for a very common user activity: browsing through the directory hierarchy.

(c) Since Gopher originated in a University setting, one of the goals was for departments to have the option of publishing information from their inexpensive desktop machines, and since much of the information can be presented as simple text files arranged in directories, a protocol modeled after a file system has immediate utility. Because there can be a direct mapping from the file system on the user's desktop machine to the directory structure published via the Gopher protocol, the problem of writing server software for slow desktop systems is minimized.

(d) A file system metaphor is extensible. By giving a "type" attribute to items in the pseudo-file system, it is possible to accommodate documents other than simple text documents. Complex database services can be handled as a separate type of item. A file-system metaphor does not rule out search or database-style queries for access to documents. A search-server type is also defined in this pseudo-file system. Such servers return "virtual directories" or list of documents matching user specified criteria.

2. The internet Gopher Model

A detailed BNF rendering of the internet Gopher syntax is available in the appendix... but a close reading of the appendix may not be necessary to understand the internet Gopher protocol.

In essence, the Gopher protocol consists of a client connecting to a server and sending the server a selector (a line of text, which may be empty) via a TCP connection. The server responds with a block of text terminated with a period on a line by itself, and closes the connection. No state is retained by the server between transactions with a client. The simple nature of the protocol stems from the need to implement servers and clients for the slow, smaller desktop computers (1 MB Macs and DOS machines), quickly, and efficiently.

Below is a simple example of a client/server interaction; more complex interactions are dealt with later. Assume that a "well-known" Gopher server (this may be duplicated, details are discussed later) listens at a well known port for the campus (much like a domain-name server). The only configuration information the client software retains is this server's name and port number (in this example that machine is rawBits.micro.umn.edu and the port 70). In the example below the Δ denotes the TAB character.

Client: (Opens connection to rawBits.micro.umn.edu at port 70)
Server: (Accepts connection but says nothing)
Client: `<CR><LF>` (Sends an empty line: Meaning "list what you have")
Server: (Sends a series of lines, each ending with CR LF)

```
0About internet GopherΔStuff:About usΔrawBits.micro.umn.eduΔ70
1Around the University of MinnesotaΔZ,5692,AUMΔunderdog.micro.umn.eduΔ70
1Microcomputer News & PricesΔPrices/Δpserver.bookstore.umn.eduΔ70
1Courses, Schedules, CalendarsΔΔevents.ais.umn.eduΔ120
1Student-Staff DirectoriesΔΔuinfo.ais.umn.eduΔ70
1Departmental PublicationsΔStuff:DP:ΔrawBits.micro.umn.eduΔ70
```

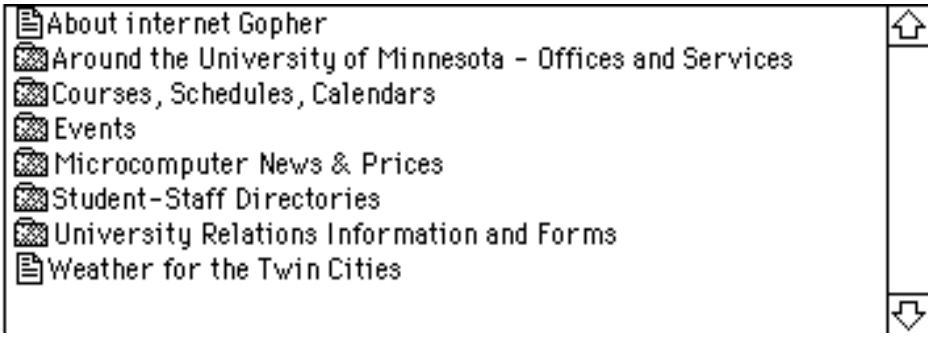
(.....etc.....)
. (Period on a line by itself)
(Server closes connection)

The first character on each line tells whether the line describes a document, directory, CSO (qi) server, or error (characters '0', '1', '2', or '3'; there are a handful more of these characters described later). The succeeding characters up to the tab form a *user display string* to be shown to the user for use in selecting this document (or directory) for retrieval. The first character of the line is really defining the type of item described on this line. In nearly every case, the Gopher client software will give the users some sort of idea about what type of item this is (by displaying an icon, a short text tag, or the like).

The characters following the tab, up to the next tab form a *selector string* that the client software must send to the server to retrieve the document (or obtain the directory listing of the directory). The selector string should mean nothing to the client software; it should never be modified by the client. In practice, the selector string is often a pathname or other file selector used by the server to locate the item desired. The last two tab delimited fields

denote the domain-name of the host that has this document (or directory), and the port at which to connect.

In the example, line 1 describes a document the user will see as “About internet Gopher”. To retrieve this document, the client software must send the retrieval string: “Stuff:About us” to rawBits.micro.umn.edu at port 70. If the client does this, the server will respond with the contents of the document, terminated by a period on a line by itself. A client might present the user with a view of the world something like the following window:



The user does not know or care that the items up for selection may reside on many different machines anywhere on the Internet.

Suppose the user selects the line “Microcomputer News & Prices”. This appears to be a directory, and so the user expects to see contents of the directory upon request that it be fetched. The following lines illustrate the ensuing client-server interaction:

Client: (Connects to pserver.bookstore.umn.edu at port 70)
Server: (Accepts connection but says nothing)
Client: `Prices/` (Sends the magic string terminated by CRLF)
Server: (Sends a series of lines, each ending with CR LF)

```
0About Prices∆Prices/Aboutus∆pserver.bookstore.umn.edu∆70
0Macintosh Prices∆Prices/Mac∆pserver.bookstore.umn.edu∆70
0ZEOS Prices∆Prices/ZEOS∆pserver.bookstore.umn.edu∆70
0IBM Prices∆Prices/Ick∆pserver.bookstore.umn.edu∆70
0Printer & Peripheral Prices∆Prices/PPP∆pserver.bookstore.umn.edu∆70
```

(.....etc.....)
. (Period on a line by itself)
(Server closes connection)

3. More details

3.1 Locating services

Documents (or other services that may be viewed ultimately as documents, such as a student-staff phonebook) are linked to the machine they are on by the trio of selector string, machine domain-name, and IP port. It is anticipated that there will be one well-known top-level or root server for an institution or campus. The information on this server may be duplicated by one or more other servers to avoid a single point of failure and to spread the load over several servers. Departments that wish to put up their own departmental servers need to register the machine name and port with the administrators of the top-level Gopher server, much the same way as they register a machine name with the campus domain-name server. An entry which points to the departmental server will then be made at the top level server. This ensures that users will be able to navigate their way down what amounts to a virtual hierarchical file system with a well known root to any campus server if they desire.

Note that there is no requirement that a department register secondary servers with the central top-level server; they may just place a link to the secondary servers in their own primary servers. They may indeed place links to any servers they desire in their own server, thus creating a customized view of thethe Gopher information universe; links can of course point back at the top-level server. The virtual (networked) file system is therefore an arbitrary graph structure and not necessarily a rooted tree. The top-level (duplicated) node is merely one convenient, well-known point of entry.

3.2 Server portability and naming

It is recommended that all registered servers have alias names that are used by Gopher clients to locate them. Links to these servers should use these alias names rather than the primary names. If information needs to be moved from one machine to another, a simple change of domain name system alias names allows this to occur without any reconfiguration of clients in the field. In short, the domain name system may simply be used in the near term to re-map a server to a new address. There is nothing to prevent secondary servers or services from running on otherwise named servers or ports other than 70[1] , however these should be reachable via a primary server.

3.3 Contacting server administrators

It is recommended that every server administrator have a document called “About internet Gopher” as the first item in their server’s top level directory. In this document should be a short description of what the server holds, as well as name, address, phone, and an e-mail address of the person who administers the server. This provides a way for users to get immediate word to the administrator of a server that is not running correctly. It is also recommended that administrators place the date of last update in files for which such information matters to the users.

[1]Port 70 has been officially allocated for internet Gopher.

3.4 Modular addition of services

The first character of each line in a server-supplied directory listing indicates whether the item is a file (character '0'), a directory (character '1'), or an error (character '3'). This is the base set of item types in the Gopher protocol. It is desirable for clients to be able to use different services and speak different protocols (simple ones such as finger; others such as CSO (qi) phonebook service, or Telnet, or X.500 directory service) as needs dictate. For example if a server-supplied directory listing marks a certain item with type character '2', then it means that to use this item, the client must speak the CSO (qi) protocol. This removes the need to be able to anticipate all future needs and hard-wire them in the basic internet Gopher protocol; it keeps the basic protocol extremely simple. In spite of this simplicity, the scheme has the capability to expand and change with the times by simply adding an agreed upon type-character for a new service. This also allows the client implementations to evolve in a modular fashion, simply by dropping in a module (or launching a new process) for some new service. The servers for the new service of course have to know nothing about internet Gopher; they can just be off-the shelf CSO, X.500, or other servers. We do not however, encourage arbitrary or machine-specific proliferation of service types.

On the other hand, subsets of other document retrieval schemes may be mapped onto the Gopher protocol by means of "gateway-servers". Examples of such servers include Gopher-to-FTP gateways, Gopher-to-Archie gateways, Gopher-to-WAIS gateways, etc. There are a number of advantages of such mechanisms. First, a relatively powerful server machine inherits both the intelligence and work, rather than the more modest, inexpensive desktop system that typically runs client software. Clients do not have to be modified to take advantage of a new resource.

3.5 Building clients

A client simply sends the retrieval string to a server if it wants to retrieve a document or view the contents of a directory. Of course, each host may have pointers to other hosts, resulting in a "graph" (not necessarily a rooted tree) of hosts. The client software will save (or rather "stack") the locations that it has visited in search of a document. The user will therefore always be able to back out of the current location by unwinding the stack. If a client does not understand what a say, type 'B' item (not a core item) is, then it simply ignores the item in the directory listing; the user never even sees it. A service (particularly a critical one) may be duplicated on more then one server. A client unable to contact a particular server should try one of the duplicated servers if they exist. Ideally, a client should pick one of the duplicated servers at random to spread the load among servers.

3.6 Building ordinary internet Gopher servers

The retrieval string sent to the server might be a path to a file or directory. It might be the name of a script, an application or even a query that generates the document or directory returned. The server uses the string it gets up to but not including a CR-LF or a TAB, whichever comes first. Following the optional TAB is a date-time descriptor

(YYYYMMDDhhmmss).

The TAB and date descriptor parts exist for the efficiency of full-text search servers (description follows). If the TAB and date descriptor are present, the server should return only items that have been modified since the specified date-time descriptor. If the server cannot implement this By-Mod-Date filtering, it can just discard the date descriptor. For example the selector:

```
Julius Caesar<CR><LF>
```

returns a directory listing if "Julius Caesar" is a directory selector, and returns the file if "Julius Caesar" selects a file. The selector:

```
Julius Caesar<TAB>19910315000000<CR><LF>
```

if a directory selector, includes all sub-directories but only includes file names that have been modified since 15 March 1991. If a file selector, the file is only returned if it has been modified since 15 March 1991, otherwise an empty return.

All intelligence is carried by the server implementation rather than the protocol. What you build into more exotic servers is up to you. Server implementations may grow as needs dictate and time allows.

3.7 Special purpose servers

There are two special server types (beyond the normal Gopher server) also discussed below:

1. A server directory listing can point at a CSO (qi) nameserver (the server returns a first character of '2') to allow a campus student-staff phonebook lookup service. This may show up on the user's list of choices, perhaps preceded by the icon of a phone-book. If this item is selected, the client software will resort to a pure CSO nameserver protocol when it connects to the appropriate host. We expect that client support for this to be superseded soon by X.500 modules. The basic module of the client software would remain unchanged; we would need to add an X.500 module.

2. A server can also point at a "full-text search server" (returns a first character of '7'). To implement campus internet (or subnet) wide searching capability, some machines may maintain full-text indexes on the contents of text documents held by some subset of Gopher servers. A "full-text search server" responds to client requests with a list of all documents that contain (or don't contain) a one or more words. The client sends the server the selector string, a tab, and the search string (words to search for). If the selector string is empty, the client merely sends the search string. The server returns the equivalent of a directory listing for documents matching the search criteria. The words "and", "or", and "not" are reserved as Boolean operators, and expressions with Boolean operators are evaluated from left to right. Example: a client might specify the search criteria as "`salmon and spinach or asparagus`" to a full-text search server and the server will respond in the normal Gopher fashion, returning a flat list of documents that match the criteria.

The CSO addition exists for historical reasons: at time of design, the campus phone-book servers at the University of Minnesota used the CSO protocol and it seemed simplest to adapt to them. The index-server is however very much a Gopher in spirit, albeit with a slight twist in the meaning of the selector-string.

3.7.1 Building CSO-servers

A CSO Nameserver implementation for UNIX is available from Steve Dorner (anon ftp from uxa.cso.uiuc.edu). We do not anticipate implementing it on other machines.

3.7.2 Building full-text search servers

An full-text search server is a special-purpose server that knows about the internet Gopher scheme for retrieving documents. These servers maintain a full-text index of the contents of plain text documents on Gopher servers in some specified domain. A gopher full-text search server was implemented using several NeXTstations because we were able to take advantage of the full-text index/search engine built into the NeXT system software. A search server for generic UNIX systems based on the public domain WAIS search engine, is also available.

By using several index servers (rather than a monolithic index server) we are able to build and search indexes in parallel (although the client software is not aware of this). While maintaining full-text indexes of documents distributed over many machines may seem a daunting task, the task can be broken into smaller pieces (update only a portion of the indexes, search several partial indexes in parallel) so that it is manageable. By spreading this task over several small, cheap (and fast) workstations we are able to take advantage of fine-grain parallelism. Again, the client software is not aware of this. Client software only needs to know that it can send a search string to an index server and receives a list of documents that contain the words in the search string.

3.8 Item type characters

The client software decides what items are available by looking at the first character of each line in a directory listing. Augmenting this list can extend the protocol. A list of defined item-type characters follows:

0 Item is a file
1 Item is a directory
2 Item is a CSO (qi) phone-book server
3 Error
4 Item is a BinHexed Macintosh file. [Use of this type is discouraged]
5 Item is DOS binary archive of some sort. [Use of this type is discouraged]
6 Item is a UNIX uuencoded file. [Use of this type is discouraged]
7 Item is an Index-Search server.
8 Item points to a text-based telnet session.
9 Item is a binary file! Client must read until the connection closes. Beware.
+ Item is a redundant server (same information as the previous server)

Characters ‘0’ through ‘Z’ are reserved. Local experiments should use other characters. We discourage arbitrary, machine-specific extensions. Note that for type 5 or type 9 the client must be prepared to read until the connection closes. There will be no period at the end of the file; the contents of these files are binary and the client must decide what to do with them based perhaps on the .xxx extension. These binary types are experimental and largely unsatisfactory. Some binary encoding scheme should really be used. Current contenders are a simple headed block, uuencode or MIME base64 encoding... (Watch this space!)

3.9 User display strings and server selector strings

User display strings are intended to be displayed on a line on a typical screen for a user’s viewing pleasure. While many screens can accommodate 80 character lines, some space is needed to display a tag of some sort to tell the user what sort of item this is. Because of this, the user display string should be kept under 70 characters in length. Clients may truncate to a length convenient to them. Selector strings sent to the server are most easily manipulated (both by file system and server application) using short (255 byte) Pascal strings on PCs... so selector strings should be less than 255 characters in length.

4 Simplicity is intentional

As far as possible we desire any new features to be carried as new protocols that will be hidden behind new document-types. The internet Gopher philosophy is:

(a) Intelligence is held by the server. Clients have the option of being able to access new document types (different, other types of servers) by simply recognizing the document-type character. Further intelligence to be borne by the protocol should be minimized.

(b) The well-tempered server ought to send "text". Should this include tabs, formfeeds, frufru? Probably not, but rude servers will probably send them anyway. Publishers of documents will be given simple tools (filters) that will alert them if there are any funny characters in the documents they wish to publish, and give them the opportunity to strip the questionable characters out; the publisher may well refuse. Note: Type 5 or 9 items are hacks for binary file transmission that may change shortly. In these cases the server just sends the binary and then closes the connection.

(c) The well-tempered client should do something reasonable with funny characters received in text; filter them out, leave them in, whatever.

Appendix.

Paul's NQBNF (Not Quite BNF) for the Gopher Protocol.

```
Note:  This is modified BNF (as used by the Pascal people) with a few
      English modifiers thrown in.  Stuff enclosed in '{}' can be
      repeated zero or more times.  Stuff in '[]' denotes a set of
      items.  The '-' operator denotes set subtraction.
```

Directory Entity

```
CR-LF     ::= ASCII Carriage Return Character followed by Line Feed
              character.

Tab       ::= ASCII Tab character.

NUL       ::= ASCII NUL character.

UNASCII   ::= ASCII - [Tab CR-LF NUL].

Lastline  ::= '.'CR-LF.

TextBlock ::= Block of ASCII text not containing Lastline pattern.

Type      ::= UNASCII.

RedType   ::= '+'.

User_Name ::= {UNASCII}.

Selector  ::= {UNASCII}.

Host      ::= {{UNASCII - ['.']} '.'} {UNASCII - ['.']}.

Note: This is a Fully Qualified Domain Name as defined in RFC 830.
     (e.g. gopher.micro.umn.edu)  Hosts that have a CR-LF
     TAB or NUL in their name get what they deserve.

Digit     ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .

DigitSeq  ::= digit {digit}.

Port      ::= DigitSeq.

Note: Port corresponds the the TCP Port Number, its value should
     be in the range [0..65535]; port 70 is officially assigned
     to gopher.

DirEntity ::= Type User_Name Tab Selector Tab Host Tab Port CR-LF
         {RedType User_Name Tab Selector Tab Host Tab Port CR-LF}



Notes:
```

It is *highly* recommended that the User_Name field contain only printable characters, since many different clients will be using it. However if eight bit characters are used, the characters should conform with the ISO Latin1 Character Set. The length of the User displayable line should be less than 70 Characters; longer lines may not fit across some screens.

The Selector string should be no longer than 255 characters.

Menu Entity

```
Menu      ::= {DirEntity} Lastline.
```

Menu Transaction (Type 1 item)

```
C: Opens Connection
S: Accepts Connection
C: Sends Selector String
S: Sends Menu Entity
```

Connection is closed by either client or server (typically server).

Textfile Entity

```
TextFile  ::= {TextBlock} Lastline
```

Note: Lines beginning with periods must be prepended with an extra period to ensure that the transmission is not terminated early. The client should strip extra periods at the beginning of the line.

TextFile Transaction (Type 0 item)

```
C: Opens Connection.
S: Accepts connection
C: Sends Selector String.
S: Sends TextFile Entity.
```

Connection is closed by either client or server (typically server).

Note: The client should be prepared for the server closing the connection without sending the Lastline. This allows the client to use fingerd servers.

Full-Text Search Transaction (Type 7 item)

```
Word      ::= {UNASCII - ' '}
BoolOp ::= 'and' | 'or' | 'not' | SPACE
SearchStr ::= Word {{SPACE BoolOp} SPACE Word}

C: Opens Connection.
C: Sends Selector String, Tab, Search String.
```

S: Sends Menu Entity.

Note: In absence of 'and', 'or', or 'not' operators, a SPACE is regarded as an implied 'and' operator. Expression is evaluated left to right.

Binary file Transaction (Type 9 or 5 item)

C: Opens Connection.
S: Accepts connection
C: Sends Selector String.
S: Sends a binary file and closes connection when done.

Note: This subject to change... perhaps soon.

Syntactic Meaning for Directory Entities

The client should interpret the type field as follows:

0 The item is a TextFile Entity.
Client should use a TextFile Transaction.

1 The item is a Menu Entity.
Client should use a Menu Transaction.

2 The information applies to a CSO phone book entity.
Client should talk CSO protocol.

3 Signals an error condition.

4 Item is a Macintosh file encoded in BINHEX format

5 Item is PC-DOS binary file of some sort. Client gets to decide.
(Note: this type is far from settled or final)

6 Item is a uuencoded file.

7 The information applies to a Index Server.
Client should use a FullText Search transaction.

8 The information applies to a Telnet session.
Connect to given host at given port. The name to login as at this host is in the selector string.

9 Item is a binary file. Client must decide what to do with it.
(Note: this type is far from settled or final)

+ The information applies to a duplicated server. The information contained within is a duplicate of the primary server. The primary server is defined as the last DirEntity that is has a non-plus "Type" field. The client should use the transaction as defined by the primary server Type field.