VISUALISING LARGE WEB APPLICATION DATASETS IN GOOGLE EARTH

By

Edward Levie

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE - COMPUTER SCIENCE

Approved:

_____

Dr. Boleslaw K. Szymanski, Thesis Adviser

_____

Dr. Sibel Adali, Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

April 2008
(For Graduation May 2008)

CONTENTS

LIST OF FIGURES

ABSTRACT

Presented herein are two methods for visualizing large geographic datasets in Google Earth. The problem is presented in the context of a typical geographic web application. Web applications are generally subject to constraints not found in other client-server system architectures. The nature of the world wide web is such that careful consideration must be made in order to facilitate rich applications such as Google Earth. Either method presented in this work can be applied to any geographic dataset.

The first method is a simple approach which makes no allowance for network conditions or topology. A geographic dataset is converted to the most basic KML which legally represents it.

RegionSplit is a more complex method but is designed to efficiently use network resources. It uses a quadtree data structure which enables the size of an average server response to be optimized. Both methods are described and compared and possible usage scenarios are mentioned.

CHAPTER 1
Introduction

1.1 Motivation

Large collections of geographic data are being created and made accessible via the internet. Researchers use this data to gain insight into past, present, and future events and to gain more thorough understanding of the world around them. One of the most effective analysis methods for this data is to visualize it on a map, possibly in combination with other data.

MetPetDB is a geochemical database for metamorphic rock samples [1, 2]. The goal of MetPetDB is to enable collaboration between researchers in the geochemical community. One of the valuable features envisioned is the display of rock samples in Google Earth. Visualizing rock samples in this manner would provide researchers with the capability to gain new insight into their collected data in concert with colleagues' data and potentially non-MetPetDB geographic data. The original motivation for the work presented in this thesis was to devise a method enabling this form of geochemical rock sample visualization. As more and more researchers make use of MetPetDB the need for efficient network utilization will grow. MetPetDB is the original intended target of this work however the methods presented are general and suitable to a variety of geographic visualization applications.

1.2 Background

Google Earth is a free application which displays an interactive virtual globe. Users can tilt, pan, and zoom the view in order to visualize cities, states, landmarks, and more in a rich 3D environment. Various layers of information can be enabled or disabled to show relative positions, concentrations, sizes, etc of combinations of data from multiple sources.

What makes Google Earth useful for this work is that its input is specified using an open modeling language called KML. This feature makes Google Earth accessible and more importantly extensible by any web application developer. Output created

by both methods in this work consists of geographic data formatted in KML.

1.3 Thesis organization

This thesis is composed of seven chapters. It starts with a brief introduction of the motivation and background of geographic data streaming and related tools. Chapter two discusses the conditions and assumptions necessary for making use of the methods herein described. The third chapter details the constructs in KML, the Keyhole Markup Language, which will be the output of each algorithm and the input to Google Earth. The following two chapters outline two methods for creating KML from a set of geographic data: chapter four describes a basic approach and chapter five presents an algorithm designed to intelligently stream data. Results, conclusions, and possible future enhancements are discussed in chapter seven. A listing of the source code used to test both methods is contained in appendix A. Appendix B contains formal specifications of the main KML elements used in this paper.

CHAPTER 2
Operating conditions

The methods described in this paper are useful given a set of operating conditions that do not apply to every geographic web application. The following assumptions are made about the conditions in which the described methods will be applied.

2.1 Dynamic content

Most projects using Google Earth serve a single common dataset to all users. Dynamic in this context refers not only to the state of the data across time but also from user to user. An example of the difference would be traffic levels versus the locations of a user's friends: in the former case the data is the same for all users while for the latter the data depends on the viewer of the data. This important difference is what drives the need for algorithms which operate on dynamic content.

2.2 Stateless connection

It is assumed that the connection between client and server is stateless. That is, the client and server treat each request as independent and store no information between requests. The world wide web and Google Earth use a stateless protocol (namely HTTP.)

Statelessness is an important attribute because it necessitates a certain amount of consideration when designing web applications. In the context of this work, the stateless nature of the communication protocol governs how queries are constructed and to what extent data can be kept current during a Google Earth session.

2.3 Computing resources

We assume that network bandwidth is more scarce than the computing resources (memory, CPU, etc) available to both the client and server. This assumption is important in that it governs the metrics by which we measure the effectiveness of

each method; network utilization is taken to be more important than the efficiency of fetching, storing, and constructing data for transmittal.

CHAPTER 3
Keyhole Markup Language

Keyhole Markup Language, or KML, is an XML-based language for describing geographic data and for controlling the behavior of a compatible viewer application (in this case Google Earth.) KML is a powerful and flexible language, but for simplicity only a small set of KML tags are described in this paper: Document, Placemark, Region, NetworkLink, NetworkLinkControl, Update, and Create. The full syntax for each of these elements is in appendix B.

In order for Google Earth to accept a KML file, it must have a content type (MIME type) of:

```
text/plain
```

or

```
application/vnd.google-earth.kml+xml
```

and have the following header:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
```

Other basic requirements also exist which Google Earth checks upon loading a KML file [4]. This header is omitted from all KML shown to reduce repetition of mundane details.

3.1 Document

A document is simply a collection of other KML elements. In each of the methods we use a single document which encompasses all of the placemarks and networklinks transmitted throughout a session.

```
<Document id="doc0">
  <name>Example Document</name>
```

```
  <!-- Placemarks, NetworkLinks, etc -->
</Document>
```

3.2 Placemark

Placemarks are the objects that we place on the earth (more precisely we use placemarks with points so they have an icon.) Where placemarks are demonstrated here, they have a position (longitude and latitude) and a name. Generally placemarks will contain descriptions, links, and images associated with their corresponding landmarks.

```
<Placemark>
  <name>Example Placemark</name>
  <Point>
    <coordinates>-73.681827, 42.730024, 0</coordinates>
  </Point>
  <!-- address, description, etc -->
</Placemark>
```

3.3 Region

Regions are bounding boxes which can be active or inactive depending on the position of the camera. Only when a region is active are the items associated with it visible to the user. That is, siblings of a region become visible when the region is active and are hidden when the region is inactive. The example given below would become active when it occupies at least 512 pixels of the view.

```
<Region>
  <LatLonAltBox>
    <north>28.382358</north> <south>21.023429</south>
    <east>13.231288</east> <west>10.298302</west>
  </LatLonAltBox>
  <Lod>
    <minLodPixels>512</minLodPixels>
```

```
  </Lod>
</Region>
```

3.4 NetworkLink and NetworkLinkControl

A networklink points Google Earth to a KML file on a network and networklinkcontrol specifies the behavior of the file that is retrieved. An important note is that the networklinkcontrol element is contained in the fetched file and not the original file loaded by Google Earth. See below for a link and a possible corresponding control.

```
<NetworkLink>
  <Link>
    <href>http://example.com/kml_link</href>
  </Link>
  <!-- Region, description, etc -->
</NetworkLink>


<NetworkLinkControl>
  <linkName>File fetched by a NetworkLink</linkName>
  <!-- Update, message, etc -->
</NetworkLinkControl>
```

3.5 Update

Updates instruct Google Earth to modify a pre-existing document or placemark. They can be used to add information, change position, or in any other way alter an existing element. The targetHref attribute points to the URL from which the existing data was loaded.

```
<Update>
  <targetHref>http://example.com/kml</targetHref>
  <!-- Change, Create, etc -->
</Update>
```

3.6 Create

Creates reference a pre-existing document and add new elements to it. In the following example the familiar structure of a document containing placemarks can be identified as the body of the create. The targetId attribute of the document refers to the existing document element in which to create new placemarks or networklinks. In this work a single document id of doc0 is used throughout for simplicity.

```
<Create>
  <Document targetId="doc0">
    <!-- Placemarks, NetworkLinks, etc -->
  </Document>
</Create>
```

CHAPTER 4
Basic method

As the name implies the basic method is the simplest method for sending data to Google Earth to be visualized. We include this straightforward approach for two reasons. First, we treat this method as a baseline to understanding the basics of representing geographic data. A minimal number of KML elements are used to demonstrate the general structure of a KML file. Second, it is quite possible that even in an application serving dynamic content the overhead incurred in implementing a more complex method may be wasteful; any given session may not produce a KML file large enough to cause concern about network utilization.

4.1 BasicKML procedure

This method produces the simplest valid KML for a given geographic dataset. The first step is to fetch all results matching the user's query parameters. It is assumed that data comes from a relational database or file system and the query specifies certain attributes of the data which should hold in order to be returned by GetResults. Each result is then made into a Placemark contained inside a single document.

```
# Input: query string
# Output: KML file
procedure BasicKML(query):
  start Document
  data = GetResults(query)
  for d in data:
    start Placemark(d.name, d.latitude, d.longitude)
    end Placemark
  end Document
```

Figure 4.1: BasicKML procedure

4.2 Output

KML generated via the basic method uses only two of the constructs previously discussed: Document and Placemark. Figure 4.2 contains KML for two placemarks. It is clear that this method simply expands the body of the document with a placemark for each result.

```
<Document id="doc0">
 <Placemark>
  <name>Taj Mahal</name>
  <Point>
   <coordinates>78.042096, 27.174885, 0</coordinates>
  </Point>
 </Placemark>
 <Placemark>
  <name>Robben Island</name>
  <Point>
   <coordinates>18.366700, -33.800000, 0</coordinates>
  </Point>
 </Placemark>
</Document>
```

Figure 4.2: Example BasicKML output for two famous landmarks

CHAPTER 5
RegionSplit method

RegionSplit and helper procedures take advantage of the spatial nature of geographic data to stream only what is important to the viewer of the data. A common data structure called a quadtree provides the basis for efficient network utilization given the operating condition assumptions from chapter 2. For brevity and readability the procedures in this chapter do not explicitly show all KML elements being output. Examining the provided examples should make clear the necessary elements to be sent to Google Earth. Furthermore, certain shortcuts are made in describing the necessary information at each stage: url is meant to be the url of the server serving the data, query is short for some set of parameters which refine the data, and box is a shortcut representing the coordinates of a bounding box on the earth. Figure 5.3 depicts the recursive call structure and responses of the RegionSplit method.

5.1 Quadtree data structure

The quadtree data structure is used to recursively partition the space occupied by a set of data so that each region contains some maximum number of elements [3]. We denote this number K. Each call to RegionSplit performs one step in the creation of the quadtree. Note that in using this method the entire quadtree is not necessarily formed; only those regions which the viewer activates are computed. Figure 5.1 illustrates a complete partitioning of a fairly sparse set of points.

5.2 RegionSplit procedure

The heart of the RegionSplit method is the following procedure of the same name. It is the algorithm which subdivides a region, specified by a bounding box, and creates either exactly four node responses or a single leaf response. The first step is to get the results matching the given query within the given bounding box. Then, depending on the size of the result set, either K or fewer placemarks (leaf) or four new regions with networklinks (nodes) are created. Divide is taken to be

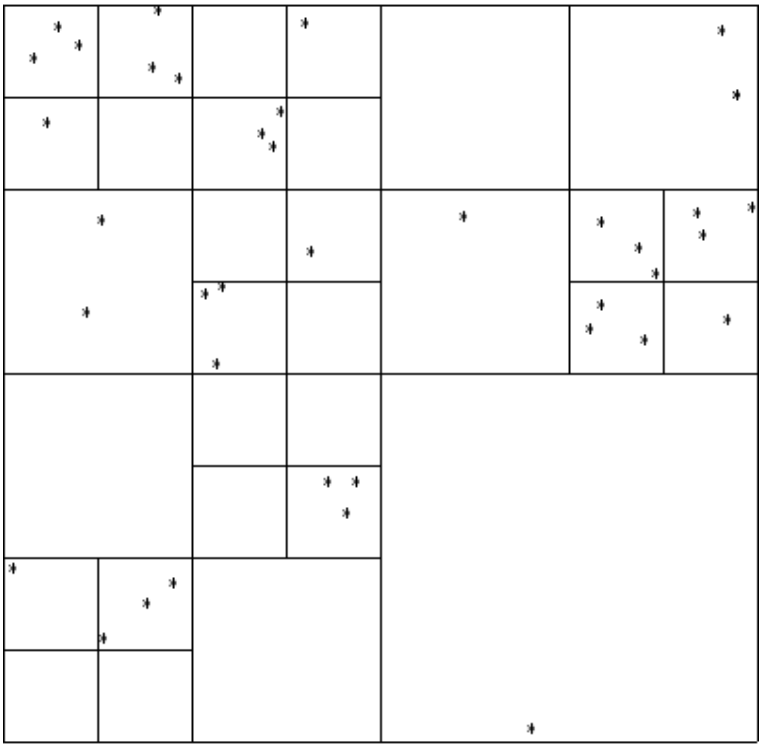

Figure 5.1: Quadtree for a random dataset with K=3

a simple procedure which returns four equal-size regions subdividing the bounding box passed into RegionSplit.

Every call to RegionSplit is passed the query, the bounding box of the current region, list of data points already sent, and the maximum number of items per region K. The query must be passed (as a GET variable) to each call because the connection is stateless. The combination of url+query is also needed so that the update creates new elements inside the original document. Sent is an array of data points that lie in the region which have already been sent. This is critical because if these are not tracked multiple copies of a result could be sent and any resending of unnecessary data hurts the network utilization the most as it is a waste of resources. Sent results are passed down the recursion tree so as to never send any data point twice. K is a parameter than can be used to limit the size of each RegionSplit response.

```
# Input: query string, bounding box, already sent data
# Output: Node or leaf response
procedure RegionSplit(query, box, sent, K):
  data = GetResults(query, box)

  # Leaf response
  if data.count() <= K:
    LeafResponse(query, data, sent)
    return

  # Node response
  regions = Divide(box)
  for r in regions:
    send up to K results from data lying inside r
    place sent results onto sent list for r
    send networklink for region
```

Figure 5.2: RegionSplit procedure

5.3 Helpers

A pair of procedures act as helpers in the RegionSplit method. The first, Load, is simply a networklink instructing Google Earth to fetch the second, Init. While a seemingly unnecessary complexity, this step is crucial as updates can only operate on documents associated with a URL and therefore loaded through a networklink. As both Load and Init are trivial procedures producing mostly static content, only their output is shown in figures 5.4 and 5.5.

5.4 Output

RegionSplit generates two types of responses. These response types are node responses and leaf responses. Figure 5.3 shows the roles of these responses as squares and triangles respectively.

5.4.1 Node response

Node responses are generated when a region in the quadtree contains more than K points. For any node response there are up to 4*K placemarks (up to K for each of the subdivided regions) and four networklink tags each containing a region.

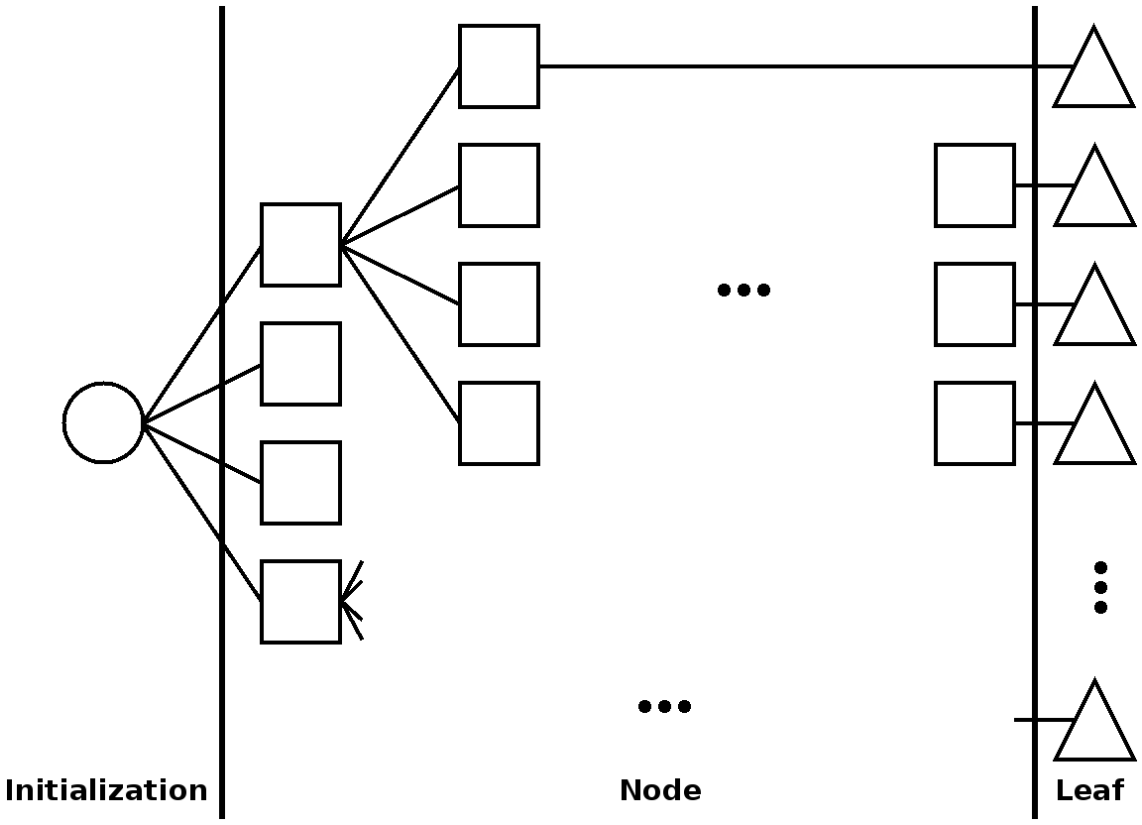

Figure 5.3: RegionSplit call sequence showing initialization, node responses, and leaf responses as a cirle, squares, and triangles respectively

Some set of up to K results are sent with each subregion so as to approximate the full result set at a lower resolution. If no results were sent with each node response the user could potentially have to zoom in to the point of a leaf in order to see a result. Such behavior would not provide a reasonable approximation of the full result set as it would not be clear where data points existed in order to zoom in for a detailed view.

When one of a node response's subregions is activated by the viewer the next level of the quadtree is requested from the server. This recursive subdivision continues until all leaf responses have been returned. Figure 5.6 provides partial KML for an example node response.

```
<NetworkLink>
  <name>Initial load</name>
  <Link>
    <href>url+query</href>
  </Link>
</NetworkLink>
```

Figure 5.4: Load output

```
<Document id="doc0">
  <NetworkLink>
    <Region>
      <LatLonAltBox>
        <north>47.67</north><south>25.39</south>
        <east>-80.16</east><west>-124.94</west>
      </LatLonAltBox>
      <Lod>
        <minLodPixels>512</minLodPixels>
      </Lod>
    </Region>
    <Link>
      <href>url+query+box(47.67,25.39,-80.16,-124.94)</href>
      <viewRefreshMode>onRegion</viewRefreshMode>
    </Link>
  </NetworkLink>
</Document>
```

Figure 5.5: Init output

5.4.2 Leaf response

Leaf responses occur when a region has no more than K points in its bounds. In such a case all of the points are sent in similar fashion to the way BasicKML formats results; no networklinks nor regions are created. The difference between a leaf and BasicKML, as evidenced in Figure 5.7, is that a leaf instructs Google Earth to update a document while BasicKML creates a document.

```
<NetworkLinkControl>
  <Update>
    <targetHref>url+query</targetHref>
    <Create>
      <Document targetId="doc0">
        <NetworkLink>
          <Region>
            <LatLonAltBox>
              <north>10</north><south>0</south>
              <east>10</east><west>0</west>
            </LatLonAltBox>
            <Lod>
              <minLodPixels>512</minLodPixels>
            </Lod>
          </Region>
          <Link>
            <href>url+query+sent+box(10,0,10,0)</href>
            <viewRefreshMode>onRegion</viewRefreshMode>
          </Link>
        </NetworkLink>
        <!-- Placemark_1, ..., Placemark_K -->

        <!-- Three other NetworkLinks -->
        <!-- ... -->

      </Document>
    </Create>
  </Update>
</NetworkLinkControl>
```

Figure 5.6: Example node response showing a single NetworkLink and Region

```
<NetworkLinkControl>
  <Update>
    <targetHref>url+query</targetHref>
    <Create>
      <Document targetId="doc0">
        <Placemark>
          <name>Taj Mahal</name>
          <Point>
            <coordinates>78.0420, 27.1748, 0</coordinates>
          </Point>
        </Placemark>
        <Placemark>
          <name>Robben Island</name>
          <Point>
            <coordinates>18.3667, -33.8000, 0</coordinates>
          </Point>
        </Placemark>
      </Document>
    </Create>
  </Update>
</NetworkLinkControl>
```

Figure 5.7: Example leaf response

CHAPTER 6
Results and Conclusions

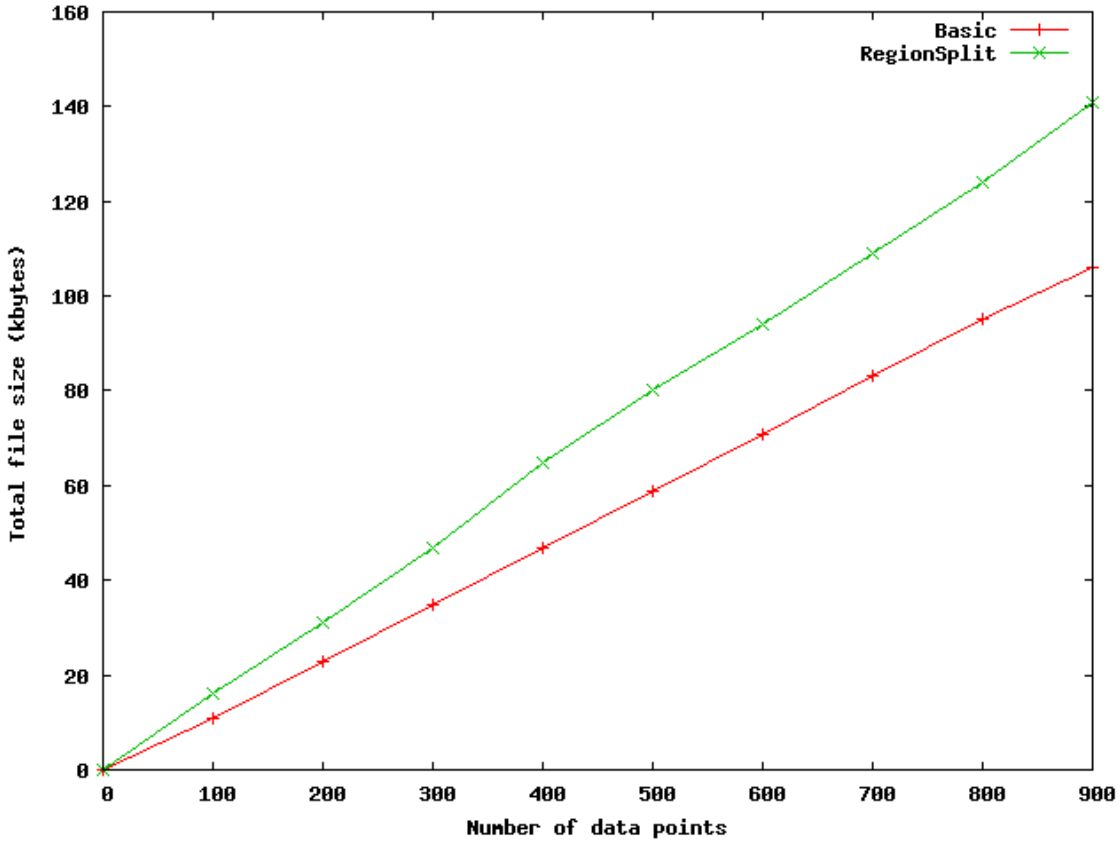

Figure 6.1: File size versus number of points in a session with K=10

Figure 6.1 compares the total amount of data transmitted throughout an entire session. RegionSplit sends around 35% more data in total than the basic method. This is expected, as overhead is incurred in sending the additional networklink, networklinkcontrol, update, and create elements for each response. Clearly RegionSplit is not significantly outperformed by the basic approach in terms of total file size.

RegionSplit is more adaptable than the basic approach due to the parameter K, the maximum number of results per region. Figure 6.2 shows that low values of K result in larger total file sizes. When K reaches approximately the total number of points divided by 4, the total file size reaches a minimum because the first call to

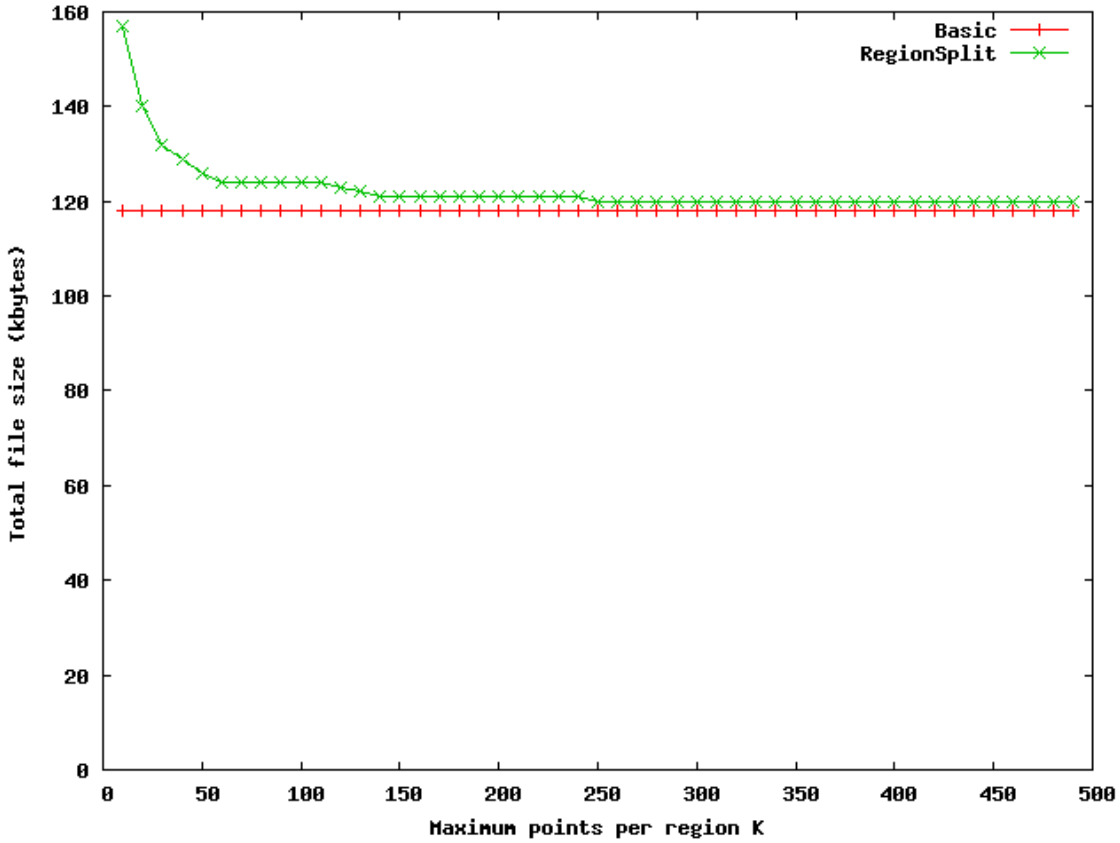

Figure 6.2: File size versus K for 1000 data points

RegionSplit results in all leaf nodes; no overhead is caused by multiple requests.

When considering efficient network utilization, oftentimes the important metric is the size of an average request [5]. Figure 6.3 illustrates the behavior of each method for a wide range of data sets. By fixing K it is possible to effectively throttle the size of the largest response using RegionSplit. The basic method operation does not depend on K so no option exists to tune it for a given network topology. Here RegionSplit boasts a significant advantage over the basic technique.

Figure 6.4 presents interesting insight into the behavior of RegionSplit. For values of K up to about 10% of the number of data points the average response size gradually increases. Between 10% and 25% there is a gradual decline to a steady state. This peak around 10% occurs where many node responses contain close to K data points but are not leaf responses. Also evident is again the fact that the basic method does not have any facility for adapting to network conditions.

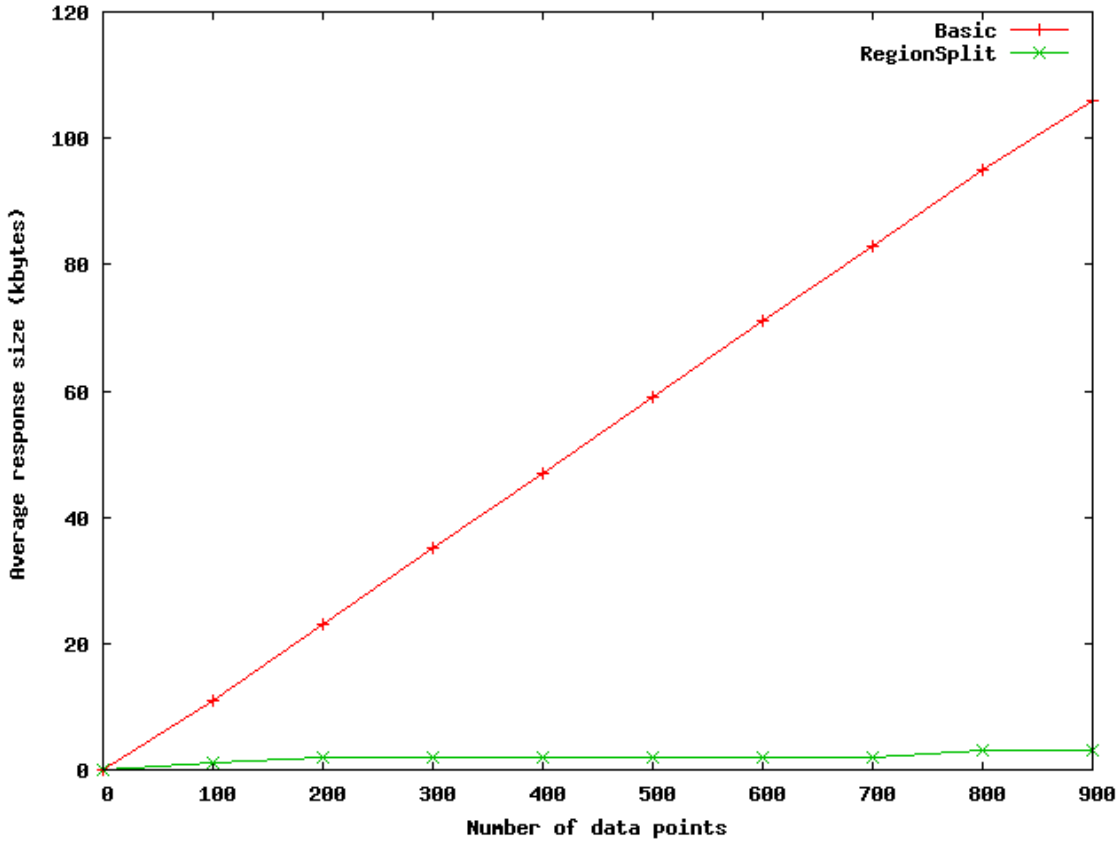

Figure 6.3: Average response size versus number of points with K=10

Having run both methods on random test data sets, it is clear that both methods can be useful. Due to its parameterization, RegionSplit is a more versatile and tunable solution. Our basic approach is clearly inflexible but not necessarily the wrong solution in all cases. When no session is expected to serve a large number of data points, the basic method provides the lowest network utilization possible. RegionSplit has proven to be adaptable to a number of situations; carefully choosing K can target a certain average response size. Analagous to the MTU, or maximum transmission unit, optimal average response size can vary widely across networks.

We have presented two methods for visualizing geographic data in Google Earth. Both methods satisfy constraints of a stateless connection, dynamic content, and limited network resources. When the number of expected results is low, it is likely most reasonable to employ the basic method. If network resources are sufficiently limited and/or the number of expected results for a given session is

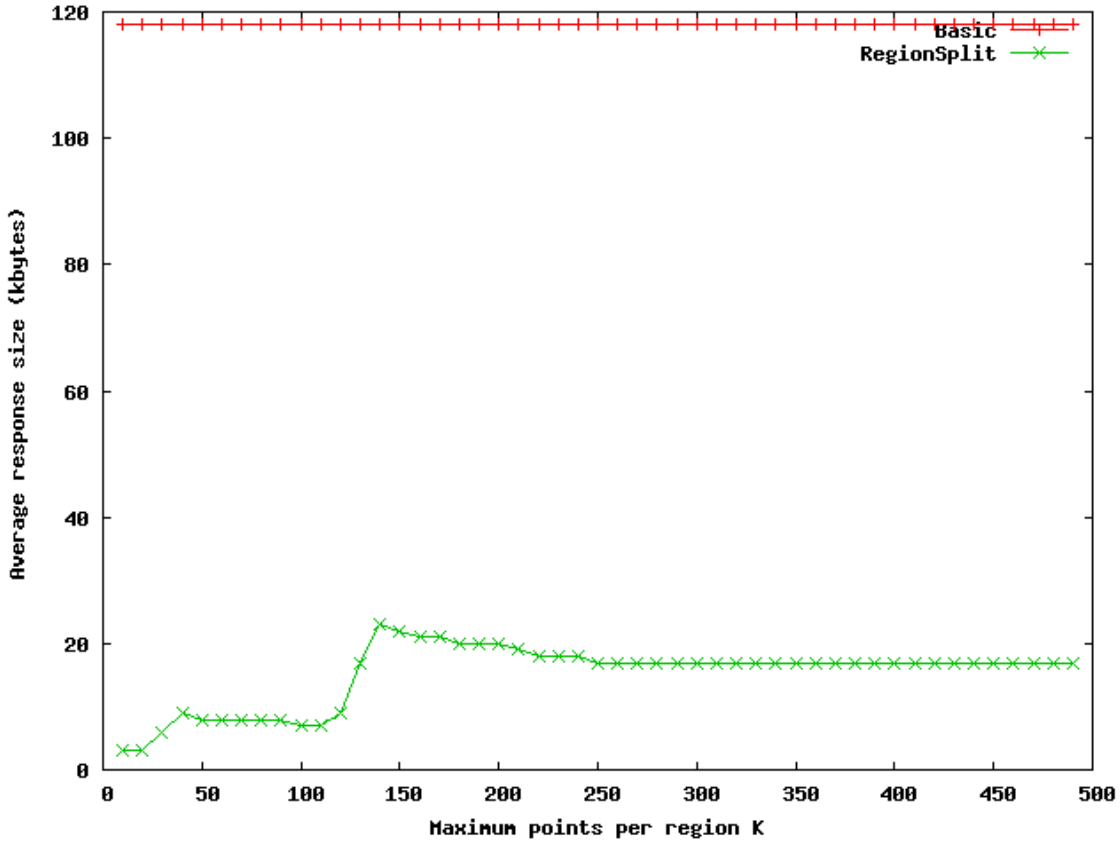

Figure 6.4: Average response size versus K for 1000 data points

highly variable, the RegionSplit method proves to be a better, but more complex, solution.

6.1 Further Work

Most notably, the RegionSplit method could be implemented in the MetPetDB framework. At the time of this writing development of MetPetDB was not at a point where Google Earth visualizations would be fruitful. It is expected that RegionSplit will be integrated into MetPetDB upon its completion.

Already a number of possible improvements to RegionSplit can be imagined. It should be possible to eliminate a small number of server requests by altering RegionSplit to create node/leaf response hybrids. When a subregion no remaining unsent results in its bounds, this could be detected and a networklink would not be created. This would eliminate the case where a networklinkcontrol is fetched

containing no placemarks. An extension of this idea would be to have a threshold c whereby a subregion containing no more than K+c results would immediately send all results and eliminate fetching c results in a separate request. This enhancement was not made in this work so as to maintain a simple RegionSplit procedure.

Various techniques exist for detecting network speed between two hosts such as that by Seshan [6] and by Wolski [7]. An extension could be investigated to automatically vary K in order to most effectively use network resources. As previously discussed, overhead incurred per request is inversely proportional to the maximum number of elements per region so tuning this paramater on the fly could help to make most efficient use of the network. In our experience the user experience is better when more results are displayed per update. Automatically tuning K would allow for increased network efficiency as well as a better user experience.

REFERENCES

[1] MetPetDB: A database for metamorphic geochemistry F.S. Spear, J.M. Pyle, S. Adali, B. K. Szymanski, A. Waters, Z. Linder, C. Ozcalar, S. O. Pearce accepted, G-Cubed (American Geophysical Union electronic journal), 2008.

[2] METPETDB: the Unique Aspects of Metamorphic Geochemical Data and Their Influence on Data Model, User Interface and Collaborations, J. Pyle, F.M, Spear, S. Adali, B.K. Szymanski, S. Pearce, A. Waters, Z. Linder, and C. Ozcaglar, 2007 GSA Denver Annual Meeting (28-31 October 2007) Geological Society of America Abstracts with Programs, Vol. 39, No. 6

[3] H. Samet 1984. The quadtree and related hierarchical data structure. ACM Comput. Surv. 16, 2, 187260.

[4] KML Tutorial, http://code.google.com/apis/kml/documentation/kml_tut.html

[5] J. Cavanaugh, Protocol overhead in IP/ATM networks, Minnesota Supercomputer Center, Inc., August 1994.

[6] S. Seshan, M. Stemm, and R. H. Katz, SPAND: Shared passive network performance discovery, in Proc. 1st Usenix Symp. Internet Technologies Systems Monterey, CA, Dec. 1997, pp. 135-146.

[7] Wolski R., Dynamically forecasting network performance using the Network Weather Service, Cluster Computing, v.1 n.1, p.119-132, 1998

[8] KML 2.1 Reference, http://code.google.com/apis/kml/documentation/kml_tags_21.html

APPENDIX A
Python Source Code

```
import random

# Simple class representing a data point
class Data:
    def __init__(self, name, lon, lat, id=0):
        self.name, self.longitude, self.latitude, self.id = name, lon, lat, id

# Generate num points in the given bounds
def RandomData(num, min_lon, max_lon, min_lat, max_lat):
    data = []
    for i in range(num):
        data.append(Data('Data point name',
                         random.random() * (max_lon - min_lon) + min_lon,
                         random.random() * (max_lat - min_lat) + min_lat, i))
    return data

# Basic approach
def BasicKML(data):
    out = []
    out.append('<Document id="doc0">')
    for d in data:
        out.append('<Placemark>')
        out.append('<name>%s</name>' % d.name)
        out.append('<Point>')
        out.append('<coordinates>%f, %f, 0</coordinates>' % (d.longitude, d.latitude))
        out.append('</Point>')
        out.append('</Placemark>')
    out.append('</Document>')
    return '\n'.join(out)

# RegionSplit method helper
def Load():
    return """<NetworkLink>
<name>Initial load</name>
<Link>
<href>http://example.com?query</href>
</Link>
</NetworkLink>"""

# RegionSplit method helper
def Init(data):
    return """<Document id="doc0">
```

```
<NetworkLink>
<Region>
<LatLonAltBox>
<north>50</north><south>0</south>
<east>50</east><west>0</west>
</LatLonAltBox>
<Lod>
<minLodPixels>512</minLodPixels>
</Lod>
</Region>
<Link>
<href>http://example.com/?query&min_lon=0&max_lon=50&min_lat=0&max_lat=50</href>
<viewRefreshMode>onRegion</viewRefreshMode>
</Link>
</NetworkLink>
</Document>"""

# Removes all data points outside of the region and points already sent
def PruneData(data, min_lon, max_lon, min_lat, max_lat, sent):
    n = []
    for d in data:
        if d.longitude >= min_lon and d.longitude <= max_lon and \
                d.latitude >= min_lat and d.latitude <= max_lat:
            insent = False
            for s in sent:
                if s.id == d.id:
                    insent = True
            if not insent:
                n.append(d)
    return n

# Helper which outputs placemark tags
def Placemark(d):
    out = []
    out.append("""<Placemark>
<name>%s</name>
<Point>
<coordinates>%f, %f, 0</coordinates>
</Point>
</Placemark>""" % (d.name, d.longitude, d.latitude))
    return '\n'.join(out)

# Helper which outputs networklink and region tags
def NodeResponse(min_lon, max_lon, min_lat, max_lat, sent):
    out = []
    out.append("""<NetworkLink>
<Region>
```

```
<LatLonAltBox>
<west>%f</west><east>%f</east>
<south>%f</south><north>%f</north>
</LatLonAltBox>
<Lod>
<minLodPixels>512</minLodPixels>
</Lod>
</Region>
<Link>
<href>http://example.com/?query&sent=%s&box=%f,%f,%f,%f</href>
<viewRefreshMode>onRegion</viewRefreshMode>
</Link>
</NetworkLink>""" % (min_lon,max_lon,min_lat,max_lat,','.join(map(str,sent)),
                     min_lon,max_lon,min_lat,max_lat))
    return '\n'.join(out)

# Heart of the RegionSplit method
# In actual use the recursive RegionSplit calls would not be included in
# each call's output.  It is shown here as such to aid in gathering test data.
def RegionSplit(data, min_lon, max_lon, min_lat, max_lat, sent, K, pure=False, gather=[]):
    data = PruneData(data, min_lon, max_lon, min_lat, max_lat, sent)

    out = []
    out.append("""<NetworkLinkControl>
<Update>
<targetHref>http://example.kml?query</targetHref>
<Create>
<Document targetId="doc0">""")

    # leaf
    if len(data) <= K:
        for i in data:
            out.append(Placemark(i))
    else:   # node
        mid_lon = (min_lon + max_lon) / 2.0
        mid_lat = (min_lat + max_lat) / 2.0

        # bottom left
        tempsent = sent[:]
        tempdata = data[:]
        tempdata = PruneData(tempdata, min_lon, mid_lon, \
                             min_lat, mid_lat, tempsent)
        for i in tempdata[:K]:
            out.append("""<Placemark>
<name>%s</name>
<Point>
<coordinates>%f, %f, 0</coordinates>
```

```
</Point>
</Placemark>""" % (i.name, i.longitude, i.latitude))
            tempdata.remove(i)

        if len(tempdata) > 0:
            out.append(NodeResponse(min_lon, mid_lon, \
                                min_lat, mid_lat, tempsent))

        out.append(RegionSplit(tempdata, min_lon, mid_lon, \
                            min_lat, mid_lat, tempsent, K, pure, gather))
        if pure: out.pop()

        # top left
        tempsent = sent[:]
        tempdata = data[:]
        tempdata = PruneData(tempdata, min_lon, mid_lon, \
                           mid_lat, max_lat, tempsent)
        for i in tempdata[:K]:
            out.append(Placemark(i))
            tempdata.remove(i)

        if len(tempdata) > 0:
            out.append(NodeResponse(min_lon, mid_lon, \
                                mid_lat, max_lat, tempsent))
        out.append(RegionSplit(tempdata, min_lon, mid_lon, \
                            mid_lat, max_lat, tempsent, K, pure, gather))

        # bottom right
        tempsent = sent[:]
        tempdata = data[:]
        tempdata = PruneData(tempdata, mid_lon, max_lon, \
                           min_lat,  mid_lat, tempsent)
        for i in tempdata[:K]:
            out.append(Placemark(i))
            tempdata.remove(i)

        if len(tempdata) > 0:
            out.append(NodeResponse(mid_lon, max_lon, \
                                min_lat, mid_lat, tempsent))

        out.append(RegionSplit(tempdata, mid_lon, max_lon, \
                            min_lat, mid_lat, tempsent, K, pure, gather))

        # top right
        tempsent = sent[:]
        tempdata = data[:]
        tempdata = PruneData(tempdata, mid_lon, max_lon, mid_lat, max_lat, tempsent)
```

```
        for i in tempdata[:K]:
            out.append(Placemark(i))
            tempdata.remove(i)

        if len(tempdata) > 0:
            out.append(NodeResponse(mid_lon, max_lon, \
                                    mid_lat, max_lat, tempsent))

        out.append(RegionSplit(tempdata, mid_lon, max_lon, \
                               mid_lat, max_lat, tempsent, K, pure, gather))


    # end
    out.append("""</Document>
</Create>
</Update>
</NetworkLinkControl>""")

    if pure: gather.append(len('\n'.join(out)))
    return '\n'.join(out)

if __name__ == '__main__':
    # Example for running each method
    data = RandomData(1000, 0, 50, 0, 50)
    BasicKML(data)
    Init(data)
    Load()
    RegionSplit(data, 0, 50, 0, 50, [], 10)
```

APPENDIX B
Keyhole Markup Language Syntax

The full KML schema and specifications are available from Google [8].

```
<Document id="ID">
  <!-- inherited from Feature element -->
  <name>...</name>                          <!-- string -->
  <visibility>1</visibility>                <!-- boolean -->
  <open>1</open>                            <!-- boolean -->
  <address>...</address>                    <!-- string -->
  <AddressDetails xmlns="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0">
    ...</AddressDetails>                    <!-- string -->
  <phoneNumber>...</phoneNumber>            <!-- string -->
  <Snippet maxLines="2">...</Snippet>       <!-- string -->
  <description>...</description>            <!-- string -->
  <LookAt>...</LookAt>
  <TimePrimitive>...</TimePrimitive>
  <styleUrl>...</styleUrl>                  <!-- anyURI -->
  <StyleSelector>...</StyleSelector>
  <Region>...</Region>
  <Metadata>...</Metadata>

  <!-- specific to Document -->
  <!-- 0 or more Schema elements -->
  <!-- 0 or more Feature elements -->
</Document>
```

Figure B.1: Document syntax

```
<Placemark id="ID">
  <!-- inherited from Feature element -->
  <name>...</name>                          <!-- string -->
  <visibility>1</visibility>                <!-- boolean -->
  <open>1</open>                            <!-- boolean -->
  <address>...</address>                    <!-- string -->
  <AddressDetails xmlns="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0">
    ...</AddressDetails>                    <!-- string -->
  <phoneNumber>...</phoneNumber>            <!-- string -->
  <Snippet maxLines="2">...</Snippet>       <!-- string -->
  <description>...</description>            <!-- string -->
  <LookAt>...</LookAt>
  <TimePrimitive>...</TimePrimitive>
  <styleUrl>...</styleUrl>                  <!-- anyURI -->
  <StyleSelector>...</StyleSelector>
  <Region>...</Region>
  <Metadata>...</Metadata>

  <!-- specific to Placemark element -->
  <Geometry>...</Geometry>
</Placemark>
```

Figure B.2: Placemark syntax

```
<Region id="ID">
  <LatLonAltBox>
    <north></north>                         <!-- required; kml:angle90 -->
    <south></south>                         <!-- required; kml:angle90 -->
    <east></east>                           <!-- required; kml:angle180 -->
    <west></west>                           <!-- required; kml:angle180 -->
    <minAltitude>0</minAltitude>            <!-- float -->
    <maxAltitude>0</maxAltitude>            <!-- float -->
    <altitudeMode>clampToGround</altitudeMode>
      <!-- kml:altitudeModeEnum: clampToGround, relativeToGround, or absolute -->
  </LatLonAltBox>
  <Lod>
    <minLodPixels>0</minLodPixels>          <!-- float -->
    <maxLodPixels>-1</maxLodPixels>         <!-- float -->
    <minFadeExtent>0</minFadeExtent>        <!-- float -->
    <maxFadeExtent>0</maxFadeExtent>        <!-- float -->
  </Lod>
</Region>
```

Figure B.3: Region syntax

```
<NetworkLink id="ID">
  <!-- inherited from Feature element -->
  <name>...</name>                          <!-- string -->
  <visibility>1</visibility>                <!-- boolean -->
  <open>1</open>                            <!-- boolean -->
  <address>...</address>                    <!-- string -->
  <AddressDetails xmlns="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0">
    ...</AddressDetails>                    <!-- string -->
  <phoneNumber>...</phoneNumber>            <!-- string -->
  <Snippet maxLines="2">...</Snippet>       <!-- string -->
  <description>...</description>            <!-- string -->
  <LookAt>...</LookAt>
  <TimePrimitive>...</TimePrimitive>
  <styleUrl>...</styleUrl>                  <!-- anyURI -->
  <StyleSelector>...</StyleSelector>
  <Region>...</Region>
  <Metadata>...</Metadata>

  <!-- specific to NetworkLink -->
  <Link>...</Link>
  <refreshVisibility>0</refreshVisibility> <!-- boolean -->
  <flyToView>0</flyToView>                  <!-- boolean -->
</NetworkLink>
```

Figure B.4: NetworkLink syntax

```
<NetworkLinkControl>
  <minRefreshPeriod>0</minRefreshPeriod>          <!-- float -->
  <cookie>...</cookie>                            <!-- string -->
  <message>...</message>                          <!-- string -->
  <linkName>...</linkName>                        <!-- string -->
  <linkDescription>...</linkDescription>          <!-- string -->
  <linkSnippet maxLines="2">...</linkSnippet>     <!-- string -->
  <expires>...</expires>                          <!-- kml:dateTime -->
  <Update>...</Update>                            <!-- Change, Create, Delete -->
  <LookAt>...</LookAt>
</NetworkLinkControl>
```

Figure B.5: NetworkLinkControl syntax

```
<Update>
  <targetHref>...<targetHref>     <!-- URL -->
  <Change>...</Change>
  <Create>...</Create>
  <Delete>...</Delete>
</Update>
```

Figure B.6: Update syntax