Home Networking

George K. Thiruvathukal

Loyola University Chicago, gkt@cs.luc.edu

Recommended Citation


This work is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 License.

Copyright © 2006 George K. Thiruvathukal
MY COEDITORS (PAUL DUBOIS AND KONSTANTIN LÄUFER) AND I ARE AMONG A GROWING NUMBER OF INDIVIDUALS WHO INSTALL AND MAINTAIN A HOME COMPUTER NETWORK. WHEN PAUL AND I FIRST DISCUSSED THE IDEA OF WRITING AN ARTICLE ABOUT THE SUBJECT SEVERAL MONTHS AGO, MY FIRST THOUGHT WAS THAT YOU—OUR CHERISHED READERS—MIGHT CONSIDER THIS WHOLE NOTION SO TRIVIAL AS TO NOT BE WORTHY OF A COLUMN THAT FOCUSES MOSTLY ON SCIENTIFIC PROGRAMMING AND SOFTWARE DEVELOPMENT TOPICS.

But based on the realization that we three computer scientists devote a significant part of our time to network and system administration, it almost goes without saying that networking is becoming more commonplace at home than at work. Although most of the ideas behind networking are relatively straightforward, several interesting technical and business decisions must be made as an ongoing part of the process. In this first installment of what I hope turns into a discussion about more advanced applications, the focus is on revealing the secrets in a typical home router/switch. (Here, a router has both switching and routing functions.)

/home/sysadmin

My piano tuner recently began experiencing some problems with his computer. The machine he uses to run his business and accounting functions also doubles as a game and peer-to-peer file-sharing system for his two kids, both of whom serve as the computer’s system administrators (sysadmins). When I spoke with him, his computer was barely functioning—whatever his kids had installed on it had rendered it almost unconscious.

I suggested that he keep his computer off-limits for the most part and get a different setup for the kids. He was a bit reluctant about the idea of buying a complete system, mainly because he had limited space in which to keep everything. I explained that he could use a KVM switch, cleverly named for its ability to switch a keyboard, video (monitor), and mouse among multiple computers electronically or with software. Although such a configuration only allows at most one person to use a computer at a time, it’s a space-saving option for the home office that would give him back some sanity. Better yet, it could be part of a network-based solution to mirror the data files zipping between the computers at a blazing 100 Mbps (Ethernet 100).

My piano tuner is still mulling over the possibilities. After talking to him, though, I realized that, for most people in the US today, the home computer setup is basically a small office/home office environment. Even those of us without small businesses have small-business-like interests. We work from home, bring work home, play from home, and occasionally sleep at home, too. In increasing numbers—even among the most technical of us—we share a common bond with my piano tuner. We need at least a subset of what we get at the office, which means we must become part-time sysadmins ourselves (unfortunately, without the pay).

Home Network

Essentially, a home network is a private Internet or intranet. The notion of an intranet became popular in the mid-to-late 1990s, when corporations wanted the benefits of networking but with the added ability to maintain a separate security domain. The Internet Engineering Task Force’s RFC 1918 (www.ietf.org/rfc/rfc1918.txt) addresses the goal of making intranets feasible by using what are known as private address classes. These classes define a set of IP addresses for intranet use and let you support as many hosts as you’re likely to ever need. Table 1 shows the presence of a nonroutable set of IPs within each address class.

Most routers are smart enough to accommodate any of the private IP ranges in Table 1, but some older routers will limit you to a class C style 192.168.X subnet with the ability to set X (this still works well for most purposes). Most users can live with the default subnet (usually 192.168.0. or 192.168.1.) because they probably won’t need to address more than 2⁸ hosts. For the record, you don’t really get to use the complete address space in a network.
anyway; you can’t use .255 to address individual hosts within the intranet, for example, because it’s part of the Internet masking strategy—that is, a network mask helps determine which part of the IP address is the network (for instance, 192.168.1.) versus the host (.N within 192.168.1.)—or the address you assign to the router (which is typically .1). You also can’t use .0, which works for the network address but not individual hosts within the network. Even so, this leaves 253 usable addresses for most home users, which is more than you’ll need unless you’re opening a supercomputing center at home.

On my home network router (Netgear WGT624), the default IP address for the router itself is 192.168.0.1. The subnet mask is 255.255.255.0, which means that all the bits from the first three components of the router’s IP address (192.168.0.) represent the network portion of the address. In practical terms, this means that addresses available for use in the private network range from 192.168.0.2 to 192.168.0.254. Addresses 192.168.0.1 and 192.168.0.255 can’t be used: the former is assigned to the router itself whereas the latter isn’t usable because it’s indistinguishable from the “mask” bits.

This addressing scheme is called private network addressing for a reason. The addresses are truly intended for private use and not as part of the global Internet addressing scheme. Home network routers provide support for network address translation (NAT; http://computer.howstuffworks.com/nat2.htm), which rewrites the source address from any originating IP packet to appear to originate from the router’s public address. This capability is also known as IP masquerading. As we’ll see later, the notion of private addressing doesn’t preclude the possibility of individual machines within the private network being reachable from the outside. It’s possible not only to go from the private Internet to the public Internet (via masquerading), but also vice versa with a technique known as port forwarding.

Configuration
At the risk of being obvious, let’s start from the beginning, even though many of you might have already set up a router at home. The first step is to connect to the router. Most routers feature an embedded Web server to enable configuration from any computer plugged into one of the router’s external Ethernet ports. Figure 1 shows a rear perspective of the router I use.

Looking from the left, we see

- a DC input for a power adapter;
- four Ethernet local-area network (LAN) ports, which represent the box’s integrated “switch” functionality;
- a wide-area network (WAN) port, which is my gateway to the world via cable or DSL modem (I can also connect to another network—in fact, a secondary use of this kind of device is to set up a private network at the office); and
- a wireless antenna (we won’t focus on wireless in this article, though).

To configure a router, you must point to its IP address, which is likely to be http://192.168.0.1 or http://192.168.1.1. Figure 2 shows something similar to what you’ll see on your router.

You’ll need to check the manual for the administrator username and password. On my box, it’s “admin,” and the password will have to be left to your imagination.

Basic Network Configuration
Once you’re connected to the router, you’ll want to check your network settings (see Figure 3).

For the most part, the act of plugging an Ethernet cable (CAT 5, 5e, or 6) into a WAN port and connecting it to a DSL router/cable modem will result in a correctly populated basic network settings form (available on every router I’ve ever seen). Nevertheless, it’s a good idea to check everything.

The most notable aspect of this form is the IP address. Assuming your router is connected to a cable or DSL modem, and the modem itself doesn’t have integrated routing functionality, you’ll probably see a public IP address in this form. (If you’re connected to another router,


**Café Dubois**

**Google’s Summer of Code**

With much less fanfare than it deserved, the folks at Google organized the Google Summer of Code (http://code.google.com/summerofcode.html). Students submitted proposals to 39 different open-source software organizations and to Google itself. Each of the accepted projects was assigned a mentor from the open-source organization, and the students received US$4,500 if they accomplished their project to the mentor’s satisfaction. On top of that, Google even paid the organizations US$500 for each student. All the code produced had to be released open-source.

Google received 8,744 applications and approved 419 projects, according to Chris DiBona, one of Google’s coordinators on this project. Students from 49 countries participated, with a roughly 80 percent success rate. My student was from Brazil and did a great job. When you realize that a project of this scale could be organized and performed entirely in cyberspace, it shows you how profoundly the world has changed.

The Summer of Code gave students real experience working with a top professional, the chance to earn (or not) a reputation with some people who matter, and gave the open-source projects free and often talented manpower to forward their projects. For Google, I suspect the major benefit is a chance to vet future employment prospects in a way an interview could never accomplish—I know that having a summer intern is the best recruitment and interview technique of all. But do the math—this cost a lot, even if Google saved all that money by not buying Skype. I hope in the end Google does well by doing good.

Speaking of which, I mentioned Skype a while back and suddenly, wham-o, a business that appears to lose money is worth billions of dollars. If you would like me to mention your unprofitable company, I accept cash, checks, or money orders, but please, no stock. If we’re going to take another trip to fantasyland, I want to be in on it this time.

**Numeric Python**

Travis Oliphant of Brigham Young University has written a third version of a numerical array facility for Python, which will replace the original Numeric Python and its almost-successful intended replacement Numarray. This new SciPy Core version combines the best of the original Numeric with good ideas from Numarray. I’ve started to work with this version and am converting my masked array (MA) module to work with it. Although not exactly upward compatible with either of its predecessors, it’s pretty close, and facilities are provided to help make old code work. Some things can just be done better now, so converting an old script is often an opportunity to make it faster and nicer.

Oliphant is making beta releases available at http://numeric.scipy.org. SciPy itself is an extensive collection of mathematical and statistical software that works with Python and its numerical extension (http://scipy.org). SciPy, of

---

**Figure 3. Network settings.** This screen shows the basic network settings for your overall connectivity to the Internet (the public IP information), which are provided by your Internet service provider.

you might see a private IP here, which is okay. The entire Internet is made possible by routers that know routers that know other routers, and so on.) If you see a private address here, it means you can also configure another router by using a Web interface. It’s best to develop an instinct for trying .1 as the address, but keep in mind that you can use any IP address to configure a router.

In my example, the Internet service provider (ISP) provides an address 24.14.197.59 via its Dynamic Host Configuration Protocol (DHCP) server. By consulting Table 1, we see that this address is public in nature, which means we can ultimately use this address to perform some lightweight serving tasks, such as running a Web server. We’ll discuss an advanced example of this shortly, but for now, let’s proceed by reviewing
course, will use the new array facility soon. The numarray team is on board with this approach, and the modernization of Numeric will have a happy ending.

I used to think that a sufficient reason for modernization was the state of the original source code, which is difficult to understand and work with. However, looking over Oliphant’s design, it’s clear now that all the changes to Python over the intervening years have opened up some great opportunities to do things better. Here’s his list of the ways in which the new facility is better than the old:

1. It has more data types (all standard C data types plus complex floats, Boolean, string, unicode, and void *).
2. It has flexible data types in which each array can have a different item size (but all elements of the same array still have the same item size).
3. Data types are true Python types contained in a hierarchy of types.
4. It has many more array methods in addition to functional counterparts.
5. Attributes are more clearly distinguished from methods (attributes are intrinsic parts of an array, so setting them changes the array itself).
6. Array scalars cover all data types that inherit from Python scalars when appropriate.
7. Arrays can be misaligned, swapped, and placed in Fortran order in memory (facilitates memory-mapped arrays).
8. Arrays can be more easily read from text files and created from buffers.
9. Arrays can be written quickly to files in text and/or binary mode.
10. Arrays inherit from big arrays that don’t define the sequence or buffer protocol, and can therefore be very large on 64-bit platforms.
11. Fancy indexing can be done on arrays using integer sequences and Boolean masks.
12. Coercion rules are altered for mixed scalar/array operations so that scalars (anything that produces a zero-dimensional array internally) won’t determine the output type in such cases.
13. When coercion is needed, temporary buffer-memory allocation is limited to a user-adjustable size.
14. Errors are handled through the IEEE floating-point status flags and there is flexibility on a per function/module/built-in level for handling these errors.
15. You can register an error callback function in Python to handle errors that are set to “call” for their error handling.
16. Ufunc `reduce`, accumulate, and `reduceat` can occur using a different type then the array type if desired (without copying the entire array).
17. Ufunc output arrays passed in can be a different type than expected from the calculation.
18. Arbitrary classes can be passed through `ufuncs` (using `_array_wrap_ and _array_priority_`).
19. `ufuncs` can be easily created from Python functions.
20. `ufuncs` have attributes to detail their behavior, including a dynamic doc string that automatically generates the calling signature.
21. Several new `ufuncs` were added (including `fexp`, `modf`, `ldexp`, `imann`, `infinity`, `isinf`, and `signbit`).
22. New types can be registered with the system so that specialized `ufunc` loops can be written for fast support of new type objects.
23. It’s C-API enhanced, so more functionality is available from extension modules.
24. It’s C-API enhanced, so array structure access can occur via macros.
25. It has new iterator objects for easy handling in C of discontinuous arrays.
26. Types have more functions associated with them (no magic function lists in the C code); any function needed is now part of the type structure.

That’s quite a list. Well done, Travis!

For Domain Name System (DNS) information, consult your ISP (www.portforward.com/networking/dns.htm). I tend to use my university’s DNS servers because they’re a heck of a lot faster than what my ISP maintains. Your first thought might be (and rightfully so) that you shouldn’t have to think about DNS servers, but, unfortunately, you do. I’ve seen several ISPs with sluggish DNS servers, which means any DNS queries you do through them (or more precisely, the ones your browser, mail client, or any other desktop application does) will be slow. I’ll discuss DNS caching a bit later, but I can’t stress enough how important proper DNS behavior on the network is to your overall performance. I’ve experienced 1.5 Mbps connections that feel as slow as 57.6 Kbps modern access, simply because hostname lookups are slower than the time it takes to transfer actual Web page content.

You can find a list of DNS servers at www.portforward.com/networking/dns.htm; I recommend choosing from this list based on your ISP and geographical area. It’s impossible to recommend a DNS server that will work for everyone, so do some experimentation and choose what’s best for your particular home network.

After getting your initial settings established, verify that you can get elsewhere on the network by pointing your browser at a university homepage or one of your favorite sites. Again, this might seem overly obvious to you, but it’s an important preliminary step to complete so that everything else makes sense.

**LAN IP Setup**

Once you’ve established that you have
If you don’t have a router with full Dynamic Host Configuration Protocol (DHCP) support, which I understand is true for many of the routers that just passed the toddler stage (two years old), you have two options:

- Get a new router. On a recent visit to CompUSA, I saw refurbished versions of the same router I own for US$35; you can get it new for as low as $50 on the Internet. I wish to stress that I’m not a sales representative for CompUSA or Netgear—you can also purchase routers from Linksys, D-Link, and other established network gear purveyors with similar capabilities.

- Technically inclined individuals (ok, you) could set up a DHCP server.

Setting up a DHCP server is fairly straightforward with Linux (www.siliconvalleyccie.com/linux-hn/dhcp.htm). Most of the major distributions provide support for the DHCP implementation from www.phystech.com/download/dhcpcd.html. The key to reproducing my router’s functionality is to set up the Linux box as a network client and assign it an unused address in the same IP address space. Let’s assume you’ll be assigning addresses dynamically in the range 192.168.0.2 to 192.168.0.200. The Linux box is at 192.168.0.201 and will serve DHCP requests.

The DHCP configuration file is called dhcpd.conf and is usually found in /etc/dhcpd.conf. Your location may vary, depending on the flavor of Linux being used. Here’s an example configuration file:

```bash
dns-update-style interim
ignore client-updates
subnet 192.168.0.0 netmask 255.255.255.0 {
  # IP numbers to assign within the subnet
  range 192.168.0.2 192.168.0.200;
  # Lease time is how long you get to keep the IP number
  default-lease-time 86400;
  max-lease-time 86400;
  # An advantage of DHCP is that you never need to remember how to configure the client interface!
  # The following 3 lines provide the router IP, broadcast address, and subnet mask—all needed to do client
  network configuration.
  option routers 192.168.0.1;
  option broadcast-address 192.168.0.255;
  option subnet-mask 255.255.255.0;
  # Most routers forward DNS requests, so you only need the router’s IP address here.
  option domain-name-servers 192.168.0.1;
  host dhcppc7 {
    hardware ethernet 00:80:77:34:af:47; fixed-address 192.168.0.4;
  }
  host athlon64 {
    hardware ethernet 00:90:4b:53:83:07; fixed-address 192.168.0.5;
  }
  # You can have as many of these host entries as desired!
}
```

Once you set up this file, you simply start the DHCP daemon, which is usually accomplished by typing “/etc/init.d/dhcpd start”. You’ll obviously want to add this script to the default run level to ensure the service starts at boot time. Otherwise, you might have some frustrated home network users.

connectivity, it’s time to think about how you want to organize your LAN. Most routers provide the capability of LAN IP address space management; Figure 4 shows how this looks on my router.

Under the LAN TCP/IP setup section, you usually have the option of setting the IP address of the router itself. This restates a point I raised earlier: the router IP need not be .1 (in fact, many corporate environments use .254). We’ll use the .1 address here because it’s a lot easier to remember.

Although my router’s geared to consumers, it provides features you’d expect in expensive commercial network solutions such as Cisco. You can set any private address as the router’s IP address, and with network masking (also called netmask), you can indicate what part of the address represents the private network address by using binary 1. Depending on which private network address range you want, you can set the mask accordingly. Think about how many hosts you’re likely to need at home before you change anything, though. I know I’m more likely to be kicked out of my house—even if I’m the sysadmin—than I’d be allowed to keep more than a handful of computers and devices, so I’m sticking with my class C private network.

You can have any address family you want and use netmasking to subnet within the address family. Suppose, for example, I want to address approximately 2^16 hosts using a 10. address—or 10.1.* addresses. To do this, I would use 10.1.0.1 as the IP address and 255.255.0.0 as the netmask. I could use DHCP for one subnet of the 10.1 address space by starting at 10.1.1.2 and ending at 10.1.1.254; I could also go across subnets from 10.1.1.2 to 10.2.1.
254 (or even use the entire range by going to 10.255.255.254).

At this point, of course, I’d be out of my mind, but this just proves the sky’s the limit.

**IP Address Management**

Once you’ve decided how to allocate IP addresses on your network, you must complete a bit of unfinished business:

- How will you actually know what addresses are being used?
- How will you make it possible to find certain addresses permanently?

In the past, the answers to these two questions meant a decision between allocating part of your LAN’s IP address range statically (that is, via the home sysadmin) or dynamically (via DHCP). With today’s routers, the notion of assigning addresses statically is becoming a thing of the past. The best approach is to plug the device into the network and use DHCP to assign its initial, and ultimately permanent, address. If this capability isn’t available on your router, see the “Life without Full DHCP Support” sidebar.

Earlier, I used Figure 4 to show LAN IP management, but I avoided talking about address reservation, which is a capability the DHCP protocol provides (www.faqs.org/rfc/rfc2131.html).

Whenever a network interface “demands” an IP address from DHCP, its media access control (MAC) address should be used to determine its actual IP address. Sometimes the LAN IP setup isn’t the configuration screen in which you perform address reservation—instead, you might need to read the manual. Nevertheless, most routers today give you the ability to determine what devices are attached to your network (see Figure 5).

Let’s look at a device we all know and love: the printer. At home, I have a network-compatible monochrome laser printer from Brother that gets its address via DHCP. If you take a look at the list of attached devices in Figure 5, the device called dhcppc7 is assigned the address 192.168.0.4, and its Ethernet address is 00:80:77:34:AF:47 (which is the printer’s hardware address). It’s beyond this article’s scope to cover Ethernet in detail, but suffice it to say that every Ethernet network card (including wireless LAN cards) has a 48-bit address: 24 bits are used for various vendors (each of which has a unique ID), and the remaining 24 are the actual device’s ID within the vendor ID.

Using the LAN IP address reservation section, we can lock in a particular IP address assignment by clicking the add button. My router already knows the assignments that I’ve made, so it presents a list of assigned addresses and lets me click a radio button to select any new reservations. In the worst case, this requires me to look up the Ethernet address by hand, which can be tricky for an embedded device such as a network printer, but most printers have the ability to print the settings page—assuming the printer is network compatible.

To reveal the Ethernet (or MAC) address for all attached Ethernet adapters on Linux and Windows, you can use the “ifconfig -a" and “ipconfig /all“ commands, respectively. One thing I’ve avoided discussing so far is the act of associating a “common name” with the IP address, which is similar to what’s done in a corporate network because workers tend to have more important things to do than memorize the company’s IP numbers. It’s a bit beyond this article’s scope, but the short answer is DNS. You can download and install the DNS Bind (www.isc.org/index.pl/?/sw/bind/) software and use it for resolving names internal to your network. Unfortunately, most routers don’t provide DNS capability innately for managing your own host information, which is a bit unfortunate because all the necessary information seems to be there. Most items, such as printers, can be “discovered” through Windows and Macintosh OS X, so I basically need a reserved IP assignment to make sure
the printer port doesn’t change on my wife’s computer once it’s configured.

Inbound Access

So far, I’ve focused on the intranet, but most users want to expose functionality to the outside world from time to time. For my own needs, I tend to require Secure Socket Shell (SSH) access to my Linux box and the ability to do some lightweight Web serving, meaning I need to allow inbound access to ports 22 and 80, respectively, through my router. This capability is called port forwarding; Figure 6 shows how to do it through the Port Forwarding and Port Triggering screen. (Again, your router and its screens could vary.)

This screen lets you map inbound access to a particular port to a specific IP address and port combination within your network. In my network, I have a Linux box that has a private IP address 192.168.0.9, and using this screen, I added a range of ports to be forwarded (using 22 to 22 means that only port 22 is actually mapped). Figure 7 shows the mapping of ports 22 and 80 as desired.

What have we accomplished at this point? I can now access ports 22 and 80 through the public IP address (24.14.197.59) assigned by my ISP, and voila! I can now SSH into this box. Figure 8 shows the proof.

This figure also demonstrates that I could access the box through the public IP address 24.14.197.59. Notable, however, is the output of the Linux `ifconfig` command that I mentioned earlier: the `eth0` device properties show the real IP address of 192.168.0.9 within my intranet. Figure 9 shows a browser session in which I’ve connected to http://24.14.197.59 on port 80.

Again, I’ve omitted the details of actually setting up a Web server here, but as you can see, all I did for this demonstration was run Apache Web server on the Linux box. Figure 9 is nothing more than the default page for the entire site, which isn’t all that exciting.

This is all good fun, but we have a slight problem: the IP number the ISP
gave me isn’t permanent (that is, the IP itself comes from the ISP’s DHCP server upstream, thus it’s subject to change without notice). Most ISPs don’t want to give out static IP addresses primarily because most people—even now—don’t truly need them. This is true even if you’re running a small business from home.

**Dynamic DNS**

I rarely do product placements in *CiSE*, but I can’t help myself when the company is truly providing a community service with no apparent strings attached. That company is DynDNS (www.technopagan.org/dynamic/), and it provides a free dynamic DNS (DDNS) hosting solution service to home users who need a way to reach their home setup, even when the IP address changes. Figure 10 shows how I registered my hostname with DynDNS.

To take advantage of this service, you first establish a free account and then add the DynDNS service. I haven’t shown all the details of registration because I want to focus on the capability itself for brevity’s sake.

When you bring up the new dynamic DNS host form, it asks you to choose a DDNS hostname. For the purposes of my setup, I chose “gkt.” Now here comes the unbelievable part: DynDNS lets you choose what I can only describe as a “vanity” domain. I went ahead and chose “is-a-geek.org” because there’s something about what I’m doing that just feels geeky, especially when I thought I was purchasing a simple consumer-oriented device for my home networking needs. Until I started writing this article, I was unaware that the router had built-in support for DynDNS.org (see Figure 11).

Once you’ve set up your DynDNS account, you simply go to the “Use a Dynamic DNS Service” section and pick www.DynDNS.org. You then fill in the information as I’ve done in Figure 11, in which the hostname is “gkt.is-a-geek.org” and the user/password are the same as what I used to sign up for the DynDNS service.

Your router might not provide support for DDNS for the same reasons it might not provide full support for DHCP. It’s sometimes easy to forget that these are consumer devices in which the feature set might have been limited for good reasons. Nevertheless, if you have your own Linux box at home, you can add this capability as well; see the “Life without Dynamic DNS” sidebar. I should also note that you aren’t limited to using DynDNS as your provider (see links to other providers at http://directory.google.com/Top/Computers/Software/Internet/Servers/Address_Management/Dynamic_DNS_Services/).

At this point, you can repeat the same SSH and Web exercises using SSH for gkt@gkt.is-a-geek.org and http://gkt.is-a-geek.org; I won’t recapture the proof screens here. For the Web site, I leave it to you to verify that you can reach it—I plan to keep it up and running for good!

In upcoming issues, Paul, Konstantin, and I hope to go beyond networking and address certain applications of interest. In my recent Open Source Practicum course at Loyola University Chicago, for example, one of my students demonstrated how to build a box to address quality-of-service issues (www.cs.luc.edu/users/gkt/luc/osc/projects). This is becoming a major concern as people begin to do more things at home such as personal video recording (Konstantin does this at home now with his Linux box) and voice over IP (VoIP). Using such a box, a technique known as traffic shaping can help ensure that certain network-intensive applications don’t consume all the available bandwidth, thereby rendering other applications minimally usable. I’d be interested to hear about other topics in the networking space that would be of interest to you, so that we can steer our efforts accordingly.

George K. Thiruvathukal is an associate professor of computer science at Loyola University Chicago. He has a PhD in computer science from the Illinois Institute of Technology. Thiruvathukal is a member of the IEEE Computer Society and the ACM. Contact him at gkt@cs.luc.edu.