

CSIS0234B Computer and Communication Networks (Class B)

Reading for Tutorial 6

Peeping at the network

In many cases, you have a network program or system that claims that it perform some particular functions, but you wonder how it is done. Perhaps you want to create your own implementation. Or perhaps you just have a problem that the program doesn't work as you expect, and you want to see what is the problem. One would then want to capture packets sent by the program.

1. Who do the capturing

The operating system is responsible for processing all data frames received by network cards and from serial ports, and process them to see whether it should be forwarded to the network layer, forwarded to another link, processed internally, or simply discarded. Hence the operating system sees and processes every single data frame that is received by any networking device. Due to such needs, most operating systems have interface for copying frames that it received to a user program, so that the frames can later be analysed. However, there is no standard here: every system use a different interface to allow user programs to capture frames.

As a note, for Ethernet links, the physical link is a broadcast media, so it is even possible to trap everything in the network. Normally, the network card will try to be smart: if a frame does not seem to be destined to the computer (because the frame has a destination address which is neither the address of the network card, nor a broadcast or multicast address), the frame is not sent to the computer, so the operating system will not see them. This reduces the amount of frames that must be processed by the operating system, thus giving more CPU cycles to applications. However, if an administer wants to see the traffic of the network, the Ethernet card can be set into a "promiscuous mode". In this mode, the checking we just mentioned is not performed, so the operating system will receive every frame on the network. At times this is a good way to debug network problems.

2. Selecting packets

It is usually impractical to capture and display all traffic that is going on in a broadcast link: there are simply too many of them. All tools that capture frames have facilities for filtering unnecessary packets. One of these tools, `tcpdump`, has the filtering and capturing code written as a library (`libpcap`, "packet capture library") that can be reused in other similar tools. As is mentioned above, no two operating systems use the same interface for packet capturing, so the library is also important in providing a unified programming interface for packet capturing. As a result, many programs use this library for capturing frames, and hence this syntax for filtering them. Even though `tcpdump` is completely text-based, and might not be the tool of choice for many administrators (we will use a GUI version `ethereal`), most do learn the syntax that it provides for filtering frames.

The `tcpdump` program is usually invoked like `tcpdump expr`, where `expr` is an expression specifying the capture filter. (If the filter is missing, every frame will be captured and displayed.) The expression is evaluated for each frame, and if it is "true", the frame is selected and displayed. Otherwise the frame is silently dropped. There is an unlucky fact, though: the man page is a rather poor and confusing attempt at explaining the syntax of the expression. So instead of forcing you to read the man page, we highlight the main features here.

An expression is a string consisting of one or more **primitives** combined using logical operators like `and` (or `&&`), `or` (or `|`) and `not` (or `!`). There is only two level of precedence: `not` is before `and` and `or`. Everything else is left-to-right. If you don't like the result, you can use parentheses to enclose them. Those using `tcpdump` usually put the whole expression in a pair of single quotes to prevent the shell program to interpret them. E.g.,

```
tcpdump 'ip and not host 127.0.0.1'
```

contains two primitives `ip` and `net 127`, combined in such a way that the frame is selected if the first is true and the second is false. It can also be written in many other ways, e.g.,

```
tcpdump 'ip && !(host 127.0.0.1)'
```

So it remains to explain what is a primitive. There are quite a large number of them, which we will explain only a few. All primitives work by checking whether the frame satisfy some criteria.

The basic primitive is a **relational** operation: finding some particular bytes of the frame, perform some operations to form an **arithmetic expression**, and match it against another arithmetic expression using a relational operator like `=`, `!=`, `>`, etc. Arithmetic expressions are formed by things that you would expect: composing values using `+`, `-`, `*`, `/`, bitwise `&`, and bitwise `|`; overriding precedence by parentheses. Values can be a constant, or the word `len` denoting the length of the frame, or the content of the frame. The frame content is denoted in a special syntax `proto[start:size]`, saying that you want to get the `size` byte integer starting from the `start`-th byte of the protocol layer `proto` of the frame. The `size` part may be omitted, in which case it means 1 byte. E.g., the following returns true if the frame is sent to an IP multicast address, i.e., with first byte of the IP destination field (16-th byte in the IP header) to be above 224, but the whole IP destination is not 255.255.255.255:

```
tcpdump 'ip[16] >= 224 and ip[16:4] != 0xffffffff'
```

3. Various shortcuts

This works only if you can remember or lookup the location of the required field of the frame, and even so it is rather tedious. There are various **shorthands** for writing them in a more readable way. Many of these shorthands has a common form where some **keywords** are followed by an id like `www.csis.hku.hk` or `53`. The earlier example `host 127.0.0.1` is of this form: the `host` keyword is followed by the id `127.0.0.1`. If a keyword is missing, then it is assumed to be the set of keywords in the last shorthand. E.g., `host 127.0.0.1` or `172.16.0.1` means `net 127.0.0.1` or `net 172.16.0.1`. Some of the possible keywords are:

- `host hostname`: true if the IP header has `hostname` in either the source or destination address. One can be more specific about the direction by preceding the expression by `dst`, `src` or `dst and src`. One can also insert an Ethernet protocol name (one of `ip`, `arp`, `rarp` and `ip6`) to specify what kind of protocol you are interested in. So

```
arp src host 0.0.0.0
```

will get all ARP requests¹ of with source hosts 0.0.0.0. The `hostname` can also be specified

¹As explained in the lecture, ARP (Address Resolution Protocol) is the protocol for finding which Ethernet address should be used to send packets to a particular IP address. An Ethernet frame, of Ethernet protocol 0x806, is broadcasted to the network, and the one holding the IP address will reply, in effect sending the Ethernet address to the requesting party.

as a DNS name, which will be searched through the normal process using `inet_ntoa()` and `gethostbyname()`.

- `port portnum`: true if the IP header has `portnum` as the source or destination port number. Again, `dst`, `src` or `dst and src` can be used to be specific about the source and destination. One can also insert an IP protocol name (one of `tcp` and `udp`) to specify what kind of protocol you are interested in. So

```
tcp port 53
```

will get all the DNS (port 53) traffic that are done via TCP rather than the normal UDP. The `portnum` (53) can also be specified by a name found in `/etc/services` (in this case, `nameserver`).

- `proto protonum`: true if the IP packet is for protocol `protonum`. E.g., `proto 17` will trap all UDP packets. The name can also be used, so you can write `proto UDP` instead. Most likely you won't even do that: there is an abbreviation `udp` which stands for it. The other such abbreviations include `tcp` and `icmp`.

The whole expression can be preceded by the keyword `ether`, in which case it chooses frames based on Ethernet protocol number. The only ones important for us are `0x800`, `0x806` and `0x8035` (IP, ARP and RARP respectively). You don't really need to type these numbers, as you can use the abbreviations `ip`, `arp` and `rarp` instead of the more verbose `ether proto 0x800`, `ether proto 0x806` and `ether proto 0x8035`.

- `broadcast` or `multicast`: the frame is a Ethernet broadcast or multicast. This can be preceded by the keyword `ip` (i.e., `ip broadcast` and `ip multicast`) in case you want only broadcasts for IP frames, not ARP, RARP or other (non-IP) protocols.

With all these knowledge, we know the first example `ip and not host 127.0.0.1` means an IP frame that has neither source nor destination address being the loopback address 127.0.0.1.

There are other primitives that are not as useful when peeping at an IP network, and we choose to let you view the man page when the need comes. There is one extra keyword, `net`, that is also important, but we decided not to explain it as it won't make sense until you know the Internet addressing scheme.

The source IP address is set to 0.0.0.0 only when a host is starting up, to detect whether a IP collision would occur.