Netzwerk-Sniffer mit GTK-Oberfläche

Verkehrskontrolle

Grafische Oberflächen mit GTK müssen Entwickler nicht in aufwändigem Spaghetti-Code definieren. Eine ausgefeilte Oberfläche entsteht mit dem GUI-Builder Glade per Drag & Drop. Die Beschreibung im XML-Format lesen Programme zur Laufzeit ein. So entsteht ein Netzwerk-Sniffer mit schöner Oberfläche. Michael Schilli

Wer wissen möchte, welche Rechner auf dem lokalen Netzwerk ihr Unwesen treiben, und die Anzeige auch gerne noch in einem GUI hätte, der benutzt das hier vorgestellte Skript »capture.pl«. Es schnüffelt die vorbeisausenden Pakete mit Hilfe des CPAN-Moduls »Net::Pcap« (siehe auch [2]) aus der Leitung oder aus dem drahtlosen Netzwerk, dekodiert sie, stellt fest, welche aus dem LAN kommen, und zeigt die gefundenen IP-Adressen in einem Text-View-Widget (siehe Abbildung 1).

Neue Adressen platziert es oben und baut dynamisch die Anzeige auf. Im »File«-Menü findet sich ein »Reset«-Eintrag, mit dem Anwender alle gefundenen Adressen aus der Liste löschen. Über »Quit« beendet sich das Programm.

GUI in XML

Das Skript definiert die grafische Oberfläche dabei nicht über feste Programm-Anweisungen, sondern liest zur Laufzeit eine XML-Beschreibung ein. Die erstellt der Programmierer mit dem Tool Glade 2 von [3]. Anschließend baut »capture.pl« die Oberfläche auf und verarbeitet ankommende Events.

Dieser Ansatz unterscheidet sich von typischen GUI-Buildern: Sie lassen den Entwickler zwar auch mit Drag&Drop Widgets platzieren und Events definieren, wandeln das Ergebnis jedoch in Code um, der anschließend vom Entwickler noch den letzten Schliff erhält. Einmal von Hand nachbearbeiteter Code ist aber meist nicht mehr vom GUI-Builder einlesbar.

Drag & Drop

Glade beherrscht beide Verfahren, es generiert wahlweise C-Code oder eben eine XML-Beschreibung, die Programme mit der Bibliothek Libglade verarbeiten. »Gtk2::GladeXML« vom CPAN ist der zugehörige Perl-Wrapper. Abbildung 2 zeigt Glade im Einsatz. Links oben ist das Hauptfenster, mit dem Anwender ein neues Projekt anlegen. Der Werkzeugkasten unten links enthält die verschiedenen Widgets, in der Mitte ist die fertige Applikation zu sehen. Das Fenster rechts oben kümmert sich um die Eigenschaften des ausgewählten Widget wie Name, Maße, Editierbarkeit und verarbeitete Signale. Rechts unten befindet sich das Widget-Tree-Fenster, das eine hierarchische Ansicht aller erstellten Widgets in der Applikation bietet.

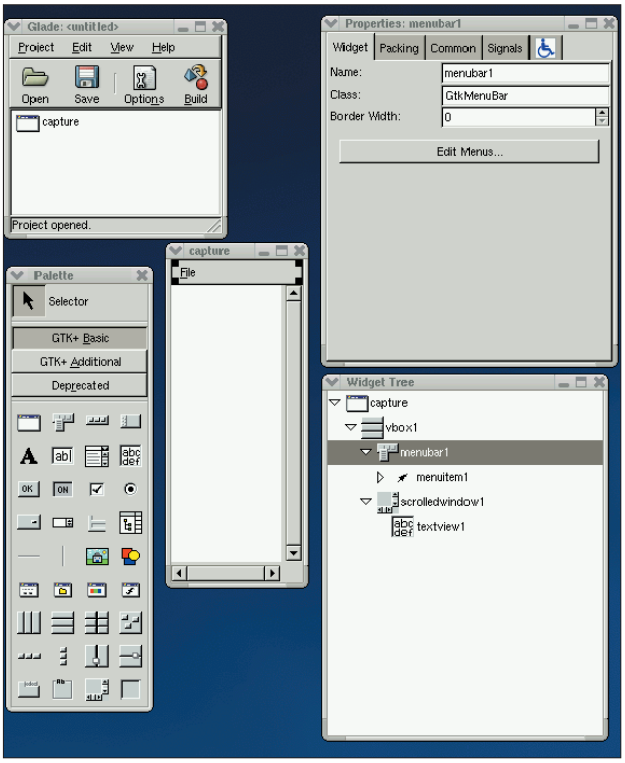
Attribute anpassen

Um eine neue GUI-Beschreibung zu erzeugen genügt ein Klick auf das Hauptfenster-Icon im Werkzeugkasten (links oben mit blauem Streifen). Daraufhin öffnet sich das leere Applikationsfenster von Abbildung 3. Ein VBox-Container sorgt dafür, dass der Menübalken oben schwebt und das Textfeld unten. Um weitere Widgets zu platzieren, genügt ein Klick auf das entsprechende Feld im Werkzeugkasten und dann ein weiterer in das Applikationsfenster.

Fehlen nur noch ein Menü, ein Scrolled Window und das Text-View-Widget. Abbildung 4 zeigt das fast fertige Applikationsfenster. Das vorher eingefügte Menü ist für die Zwecke von »capture.pl« aber noch zu umfangreich. Ein Klick auf »Edit Menus« im Properties-Fenster öffnet einen Dialog, in dem sich die Einträge bequem editieren lassen (siehe Abbildung 5). Weitere Anpassungen gehen ebenso leicht von der Hand. Länge und Breite des Hauptfensters im »capture.pl«-GUI sind über die Properties auf 300 und 120 gesetzt.

Ein Klick auf den »Save«-Knopf im Hauptfenster von Glade sichert in diesem Beispiel nach Angabe des Projektnamens »capture« zwei Dateien: »cap-

▲ **Abbildung 1:** Das GTK-Programm »capture.pl« zeigt alle Rechner im LAN in einer Liste an.



▶ **Abbildung 2:** Mit Glade konstruieren Programmierer komfortabel ihre GUIs. Die Beschreibung landet in einer XML-Datei.

ture.glade« und »capture.pglade«. Die zweite ist in diesem Zusammenhang irrelevant, in der ersten steht die XML-Beschreibung des GUI.

Das Skript »capture.pl« liest diese Beschreibung in Zeile 24 beim Aufruf des Konstruktors von »Gtk2::GladeXML« ein. Im XML stehen Definitionen der einzelnen Widgets, deren Platzierung relativ zum GUI und ihre eingestellten Attribute. So sind beispielsweise in der XML-Beschreibung zum Text-View-Widget folgende Attribute definiert:

```
<property name="editable">False</property>
<property name="cursor_visible">False
</property>
```

Es ist zu sehen, dass der Programmierer in Glade das Widget mit entsprechenden Knöpfen als nicht-editierbar und mit unsichtbarem Text-Cursor definiert hat. Die folgenden zwei Codezeilen führen zum gleichen Ergebnis:

```
$text->set_editable(0);
$text->set_cursor_visible(0);
```

Signale

Den dynamischen Teil des statisch definierten GUI legt die Methode »signal_autoconnect_all« in Zeile 47 fest. Sie verbindet die in der XML-Beschreibung mit den Widgets assoziierten Signale wie etwa »on_quit1_activate« (»File | Quit«-Menü-Eintrag angeklickt) und »on_reset1_activate« (»File | Reset«-Eintrag ausgewählt) mit entsprechenden Perl-Funktionen.

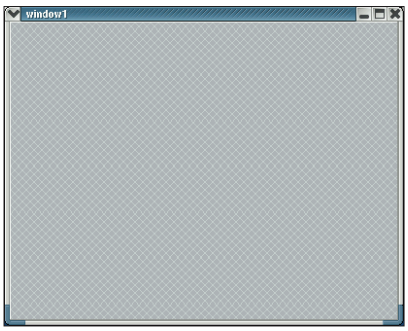


Abbildung 3: Das leere Hauptfenster in Glade, das später die Widgets enthält.

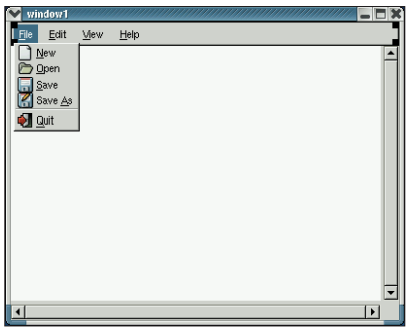


Abbildung 4: Die Oberfläche ist fast fertig. Nur die überflüssigen Menü-Einträge müssen noch raus.

Die gewählten Namen entsprechen den von Glade automatisch zugeordneten (siehe Abbildung 5). Wer möchte, ändert sie während der GUI-Konstruktion. Wenn »capture.pl« in Zeile 67 dann mit »main_loop()« in die Hauptschleife springt, geht alles seinen programmierten Gang: Das GUI erscheint auf dem Bildschirm und der Benutzer klickt nach Belieben darin herum.

Ruckelfreie Anzeige

Das Schnüffeln im Netzwerk beschlagnahmt die CPU, die sich währenddessen nicht mehr um die Oberfläche kümmert. Um also gleichzeitig das GUI in Schuss zu halten und mit dem Perl-Modul »Net::Pcap« das Ethernet abzuschnüffeln, erzeugt »capture.pl« in Zeile 32 mittels »fork()« einen Kindprozess.

Eine vorher mit »pipe()« erzeugte Pipe dient als Kommunikationskanal zwischen dem Abkömmling und dem Elternteil. Findet das Kind eine neue IP-Adresse im Netz, sendet es sie als String über das »WRITEHANDLE« der Pipe an den elterlichen GUI-Verwalter, der die Nachricht am anderen Ende der Röhre über »READHANDLE« aufschnappt.

Damit die grafische Oberfläche nicht aktiv auf Ereignisse aus der Pipe warten muss, sondern sich um Benutzereingaben kümmern kann, ist ab Zeile 62 mit

```
Glib::IO->add_watch(
    fileno(READHANDLE),
    'in', \&watch_callback);
```

ein Watch definiert, das immer dann die ab Zeile 70 definierte Callback-Funktion »watch_callback« aufruft, wenn Daten auf »READHANDLE« eintrudeln.

GTK 2 baut auf der Bibliothek Glib auf und ist daher in der Lage, deren Low-level-Dienste in Anspruch zu nehmen. Da »add_watch()« einen File-Deskriptor und kein File-Handle erwartet, wandelt die Perl-Funktion »fileno()« das Handle »READHANDLE« entsprechend um.

Im Büro des Schnüfflers

»Net::Pcap« vom CPAN ist eine Schnittstelle zur Libpcap-Bibliothek. Sie sammelt Pakete aus dem Netzwerk, analysiert und filtert sie performant nach voreingestellten Bedingungen. Die Sniffer-

Programme wie Ethereal basieren ebenfalls auf der Libpcap.

Die Funktion »snooper()« ab Zeile 90 sucht mit »Net::Pcap::lookupdev()« das erste aktive Netzwerk-Interface des Host heraus, üblicherweise »eth0«. Das darauf folgende »Net::Pcap::lookupnet()« findet die zugehörige Netzwerkadresse und -maske. »Net::Pcap::open_live()« ab Zeile 102 öffnet ein Live-Capture und schnappt sich bis zu 1024 Bytes pro Paket zur Analyse. Weil der dritte Parameter auf »1« steht, schaltet die Funktion die Netzwerkkarte in den Promiscuous Mode, veranlasst sie also dazu, nicht nur für sie bestimmte Pakete abzugreifen, sondern alle vorbeiflitzenden zu sammeln. »-1« als vierter Parameter bestimmt, dass kein Timeout (in Millisekunden) vorgegeben ist.

Die Funktion »Net::Pcap::loop()« ab Zeile 105 springt in eine Schleife, die jedes Mal, wenn sie ein Paket findet, die eingestellte Callback-Funktion »snooper_callback()« anspringt. Der zweite Parameter bestimmt mit »-1«, dass das Schnüffeln endlos weitergeht und nicht nach einer voreingestellten Anzahl von Paketen Feierabend ist.

Der letzte Parameter im »Net::Pcap::loop()«-Aufruf bestimmt eine Referenz auf einen Array mit Daten, die »snooper_callback()« bei jedem Aufruf als ersten Parameter bekommt. Mit »[$fd, $addr, $netmask]« stehen dort drei Werte: Ein File-Deskriptor »$fd« für die Pipe sowie die vorher ermittelte Netzwerkadresse und -maske.

Pakete analysieren

»Net::Pcap::loop« sorgt auch dafür, dass »snooper_callback()« die Header- und Content-Informationen des aktuell aufgeschnappten Pakets erhält. In »snooper_callback()« extrahiert »NetPacket::Ethernet::strip« die Ethernet-Informationen aus dem Paket. »NetPacket::IP->de-

Listing 1: »capture.pl«

```
001 #!/usr/bin/perl
002 ##########################################
003 # capture -- Gtk2 GUI observing the network
004 # Mike Schilli, 2004 (m@perlmeister.com)
005 ##########################################
006 use warnings;
007 use strict;
008
009 use Gtk2 -init;
010 use Gtk2::GladeXML;
011 use Glib;
012 use Net::Pcap;
013 use NetPacket::IP;
014 use NetPacket::Ethernet;
015 use Socket;
016
017 our @IPS = ();
018 our %IPS = ();
019
020 die "You need to be root to run this.\n" if
021     $> != 0;
022
023     # Load GUI XML description
024 my $g = Gtk2::GladeXML->new(
025     'capture.glade');
026
027     # Child/Parent communication pipe
028 pipe READHANDLE,WRITEHANDLE or
029     die "Cannot open pipe";
030
031     # Fork off a child
032 our $pid = fork();
033 die "failed to fork" unless defined $pid;
034
035 if($pid == 0) {
036         # Child, never returns
037     snooper(\*WRITEHANDLE);
038 }
039
040     # Parent, init text window
041 my $buf = Gtk2::TextBuffer->new();
042 $buf->set_text("No activity yet.\n");
043
044 my $text = $g->get_widget('textview1');
045 $text->set_buffer($buf);
046
047 $g->signal_autoconnect_all(
048     on_quit1_activate => sub {
049             # Stop snooper
050         kill('KILL', $pid);
051         wait();
052         Gtk2->main_quit;
053       },
054     on_reset1_activate => sub {
055             # Reset display
056         @IPS = ();
057         %IPS = ();
058         $buf->set_text("");
059       }
060 );
061
062 Glib::IO->add_watch(
063     fileno(READHANDLE),
064     'in', \&watch_callback);
065
066     # Enter main loop
067 Gtk2->main();
068
069 ##########################################
070 sub watch_callback {
071 ##########################################
072     chomp(my $ip = <READHANDLE>);
073
074         # Register IP if unknown
075     unshift @IPS, $ip
076        unless exists $IPS{$ip};
077     $IPS{$ip}++;
078
079     my $text = "";
080
081     $text .= "$_\n" for @IPS;
082
083     $buf->set_text($text);
084
085       # Return true to keep watch
086     1;
087 }
088
089 ##########################################
090 sub snooper {
091 ##########################################
092     my($fd) = @_;
093
094     my($err, $addr, $netmask);
095     my $dev = Net::Pcap::lookupdev(\$err);
096
097     if(Net::Pcap::lookupnet($dev, \$addr,
098                         \$netmask, \$err)) {
099         die "lookupnet on $dev failed";
100     }
101
102     my $object = Net::Pcap::open_live($dev,
103                         1024, 1, -1, \$err);
104
105     Net::Pcap::loop($object, -1,
106       \&snooper_callback,
107       [$fd, $addr, $netmask]);
108 }
109
110 ##########################################
111 sub snooper_callback {
112 ##########################################
113     my($user_data, $header, $packet) = @_;
114
115     my($fd, $addr, $netmask) = @$user_data;
116
117     my $edata =
118        NetPacket::Ethernet::strip($packet);
119
120     my $ip = NetPacket::IP->decode($edata);
121
122     if((inet_aton($ip->{src_ip}) &
123         pack('N', $netmask)) eq
124           pack('N', $addr)) {
125         syswrite($fd, "$ip->{src_ip}\n");
126     }
127 }
```

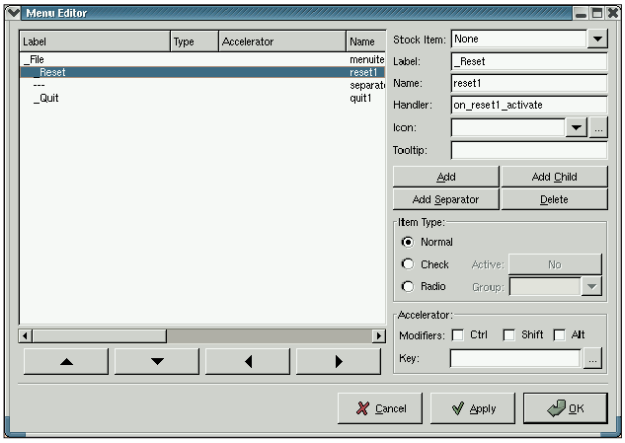


Abbildung 5: Der Menü-Editor erlaubt einfaches Editieren von Menü-Widgets.

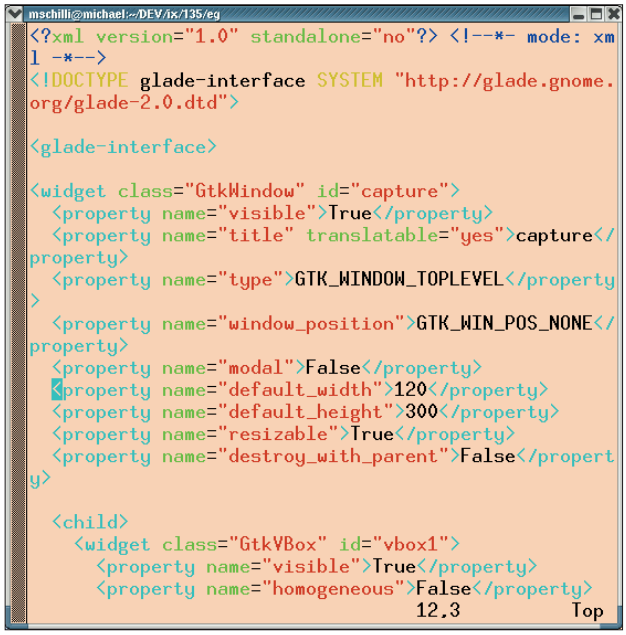


Abbildung 6: Die XML-Beschreibung der grafischen Schnittstelle in der von Glade erstellten Datei »capture.glade«.

code()« nimmt sich den IP-Layer vor und gibt eine Referenz auf einen Hash zurück, der unter dem Schlüssel »src_ip« die IP-Adresse des Senders enthält. Diesen String wandelt »inet_aton()« aus dem »Socket«-Modul ins Binärformat mit Network Byte Order um.

Die zuvor ermittelten Werte für Netzwerkadresse (»$addr«) und -maske (»$netmask«) liegen im Binärformat des Prozessors (Big oder Little Endian) vor. Der »pack«-Aufruf ab Zeile 123 wandelt sie ins maschinenunabhängige Netzwerkformat um. Danach prüft »capture.pl«, ob es die IP-Adresse »$ip« im Netzwerk »$network_addr« gibt. Ist dies erfüllt, stammt das Paket aus dem lokalen Subnetz. »syswrite()« (Zeile 125) schickt die IP-Adresse ohne Zwischenpufferung an den Elternprozess.

Diese Nachricht wandert durch die Pipe und löst im GUI wegen des in Zeile 62 aufgesetzten Watch einen Event aus, der »watch_callback()« aufruft. Dort wird der globale Array »@IPS« aufgefrischt, der alle bislang bekannten IP-Adressen enthält. Neu gefundene IPs sind noch nicht im Hash »%IPS« gespeichert und landen mit »unshift()« am Anfang des Arrays »@IPS«. Zeile 81 stellt einen Text-String mit allen bekannten IP-Adressen zusammen, die durch New-Lines getrennt sind. Zeile 83 frischt das Text-View-Widget damit auf.

Installation

Das Skript benötigt wegen der verwendeten GTK-2-Oberfläche einen ganzen Rattenschwanz an zusätzlichen Modulen. Am einfachsten ist es daher, sie mit einer CPAN-Shell zu installieren, jedoch ist manchmal ein manueller Eingriff nicht zu vermeiden. Wichtig sind vor allem die Module »ExtUtils::Depends«, »ExtUtils::PkgConfig«, »Glib«, »Gtk2«, »Gtk2::GladeXML«, »NetPacket« und »Net::Pcap«. Ist die Libglade noch nicht auf dem Rechner installiert, finden Anwender sie auf [4]. Das Programm Glade steht auf [3] zur Verfügung. Bei der Installation von »Net::Pcap« ist darauf zu achten, dass die Testphase (»make test«) als Root laufen muss, auch wenn die eigentliche Installation gar keine Root-Rechte erfordert. Tritt trotzdem ein Fehler auf, hilft es, »make install« im Build-Verzeichnis aufzurufen.

Vor dem Starten von »capture.pl« ist sicherzustellen, dass die XML-Beschreibung in »capture.glade« verfügbar ist. Wer alles unter einem Dach haben will, ändert Zeile 24:

```
my $xml = join "\n", <DATA>;
my $g = Gtk2::GladeXML->
    new_from_buffer($xml);
```

Jetzt lässt sich der XML-Salat aus »capture.glade« ans Ende von »capture.pl« in eine »DATA«-Sektion kopieren:

```
# ... Ende von capture.pl
__DATA__
<?xml version="1.0" ...
<!DOCTYPE glade-interface ...
```

Wegen des Promiscuous Mode, in den das Skript die Netzwerkkarte versetzt, muss »capture.pl« als Root laufen.

Mit Glade lassen sich nicht nur einfache Oberflächen wie diese aufbauen, sondern auch weitaus kompliziertere. Dank Drag&Drop und Wysiwyg gestaltet sich die Arbeit sehr komfortabel. Die plattformunabhängige XML-Repräsentation ist elegant und hält platzraubende statische Widget-Definitionen vom Code fern – die Programmierer konzentrieren sich auf die wesentlichen dynamischen Aspekte der Applikation. *(mwe)* ■

Infos

[1] Listings zu diesem Artikel: [ftp://ftp.linux-magazin.de/pub/listings/magazin/2004/11/Perl]
[2] Robert Casey, „Monitoring Network Traffic with Net::Pcap“: The Perl Journal 7/2004, S. 6
[3] Glade: [http://glade.gnome.org]
[4] Sourcen für Libglade: [http://ftp.gnome.org/pub/GNOME/sources/libglade]

Der Autor

Michael Schilli arbeitet als Software-Engineer für AOL/Netscape in Mountain View, Kalifornien. Er hat „Goto Perl 5“ (deutsch) und „Perl Power“ (englisch) für Addison-Wesley geschrieben und ist unter [mschilli@perlmeister.com] zu erreichen. Seine Homepage ist: [http://perlmeister.com]