lab6 实验报告

最终结果：

```
wsll@wsll-virtual-machine:~/桌面/Conputer_Network_code/minnow$ cmake --build build --target check6
Test project /home/wsll/桌面/Conputer_Network_code/minnow/build
    Start  1: compile with bug-checkers
1/3 Test  #1: compile with bug-checkers ........   Passed   11.29 sec
    Start 35: net_interface
2/3 Test #35: net_interface ....................   Passed    0.07 sec
    Start 36: router
3/3 Test #36: router ...........................   Passed    0.09 sec

100% tests passed, 0 tests failed out of 3

Total Test time (real) =  11.45 sec
Built target check6
```

代码实现：

router.hh

```
struct RouteEntry
{
  uint32_t prefix;
  uint8_t prefix_length;
  std::optional<Address> next_hop;
  size_t interface_num;
};
```

记录路由表的表项信息

```
private:
  std::vector<std::shared_ptr<NetworkInterface>> _interfaces {};
  struct TrieNode
  {
    std::unique_ptr<TrieNode> child[2]{};
    std::optional<RouteEntry> route_info{};
  };
  TrieNode _root{};

  void insert_route( const RouteEntry& entry );
  std::optional<RouteEntry> find_longest_prefix_match( uint32_t dst ) const;
};
```

记录接口信息,以及Trie树的根节点.

router.cc

```
// route_prefix: The "up-to-32-bit" IPv4 address prefix to match the datagram's destination address against
// prefix_length: For this route to be applicable, how many high-order (most-significant) bits of
//    the route_prefix will need to match the corresponding bits of the datagram's destination address?
// next_hop: The IP address of the next hop. Will be empty if the network is directly attached to the router (in
//    which case, the next hop address should be the datagram's final destination).
// interface_num: The index of the interface to send the datagram out on.
void Router::add_route( const uint32_t route_prefix,
                        const uint8_t prefix_length,
                        const optional<Address> next_hop,
                        const size_t interface_num )
{
  cerr << "DEBUG: adding route " << Address::from_ipv4_numeric( route_prefix ).ip() << "/"
       << static_cast<int>( prefix_length ) << " => " << ( next_hop.has_value() ? next_hop->ip() : "(direct)" )
       << " on interface " << interface_num << "\n";

  RouteEntry entry {
    .prefix = route_prefix,
    .prefix_length = prefix_length,
    .next_hop = next_hop,
    .interface_num = interface_num
  };
  insert_route(entry);
}
void Router::insert_route(const RouteEntry &entry) {
  TrieNode *cur = &_root;

  for (uint8_t i = 0; i < entry.prefix_length; i++) {
    uint8_t bit = (entry.prefix >> (32 - i - 1)) & 1;
    if (!cur->child[bit]) {
      cur->child[bit] = std::make_unique<TrieNode>();
    }
    cur = cur->child[bit].get();
  }
  cur->route_info = entry;
}
```

添加路由：将路由表中的路由信息添加到Trie树中

```cpp
std::optional<RouteEntry> Router::find_longest_prefix_match(uint32_t dst) const {
  const TrieNode *cur = &_root;
  std::optional<RouteEntry> best_match = std::nullopt;

  for (int i = 0; i < 32; i++) {
    if (cur->route_info.has_value()) {
      best_match = cur->route_info;
    }
    uint8_t bit = (dst >> (31 - i)) & 1;
    if (!cur->child[bit]) {
      break;
    }
    cur = cur->child[bit].get();
  }

  if (cur->route_info.has_value()) {
    best_match = cur->route_info;
  }

  return best_match;
}
```

最长前缀匹配：通过Trie树可以在O(1)的时间内找到最长前缀匹配的路由表项

```cpp
void Router::route()
{

  for (auto &iface : _interfaces) {
    auto &incoming = iface->datagrams_received();

    while (!incoming.empty()) {
      InternetDatagram datagram = std::move(incoming.front());
      incoming.pop();

      if (datagram.header.ttl <= 1) {
        continue;
      }

      datagram.header.ttl -= 1;
      datagram.header.compute_checksum();

      uint32_t dst_addr = datagram.header.dst;
      auto route_opt = find_longest_prefix_match(dst_addr);

      if (!route_opt.has_value())
      {
        continue;
      }

      const auto &route = route_opt.value();
      Address next_hop_addr = route.next_hop.has_value()? route.next_hop.value(): Address::from_ipv4_numeric(dst_addr);
      interface(route.interface_num)->send_datagram(datagram, next_hop_addr);
    }
  }
}
```

路由算法：

取出接口中的各个报文，检查ttl，如果小于1就丢弃。否则减去1，通过最长前缀匹配，找到接

口，发送过去即可。