ECEN 689– Real-Time Wireless Networks: Project 2 (Due on 4/1)

Ping-Chun Hsieh
lleyfede@tamu.edu

Tao Zhao
alick@tamu.edu

Dongni Han
handongni2015@tamu.edu

Terminology

In our report, we use "server" to denote the WiFi access point (AP), and "client" to denote the terminal device such as a mobile phone, a tablet, and so on. Throughout our simulation, we let node 0 be the server, and node 1 be the client, which correspond to the device A and B in the problem descriptions respectively.

S-WiFi is the name of our application as well as our project. It stands for Smart WiFi, or whatever you think it is.

Simulation Setup

Table 1: Parameters of the wireless channel.

Item	Value
Path loss exponent	2.0
Shadowing deviation	4.0 dB
Reference distance	1.0 m

Throughout the report, we consider a single wireless link between two devices, say A and B. The transmitter power level is 10 mW. We use the shadowing module as the wireless channel. The parameters of the channel are summarized in Table 1.

Table 2: Parameters of the 802.11b MAC.

Item	Value
Data rate	11 Mb/s
Basic rate	1 Mb/s
PLCP data rate	1 Mb/s
Preamble length	144 bits
Slot time	20 μs
SIFS	10 μs

For the medium access control (MAC) layer, we use the 802.11 module built in ns-2. Following the IEEE 802.11b standard, the MAC layer parameters are chosen as in Table 2.

Uplink Transmissions with PCF

1 Baseline Policy

In PCF mode, the AP decides which client can transmit: The AP will first send a POLL packet to the selected client. A client can only transmit its packet after it receives the POLL packet from the AP. This allows the AP to have full control over which client transmits. However, there is one problem: Packets arrive at the clients, and the AP cannot know which client has packets for transmission. Baseline policy: At the beginning of each interval, the AP asks each client, one by one, the number of packets that it generates. After this process, the AP also knows the number of packets at each client, and it can make the best decision.

First of all, the server can poll packets or poll data implemented by function `command` in the swifi.cc file. Then, the client enables to send number of packets or data to the server implemented by function `recv` in the swifi.cc file. Also, we need function `scheduleRoundRobin` to let AP ask each client one by one the number of packets that it generates at the beginning of each interval. Then, we need function `scheduleMaxWeight` to implement maxweight policy.

Besides, we define four packet types.

Table 3: Data Type

Data Type	Meaning
`SWiFi_POLL_NUM`	Poll number of packets in uplink
`SWiFi_PKT_POLL_DATA`	Poll data packet transmission in uplink
`SWiFi_PKT_NUM_UL`	Packet in uplink that carries number of packets at client
`SWiFi_PKT_DATA_UL`	Data packet in uplink(client to server)

What's more, four poll states are defined below.

Table 4: Poll State

Poll State	Meaning
`SWiFi_POLL_NONE`	Poll nothing
`SWiFi_POLL_NUM`	Poll num of packets
`SWiFi_POLL_DATA`	Poll data
`SWiFi_POLL_IDLE`	Idle

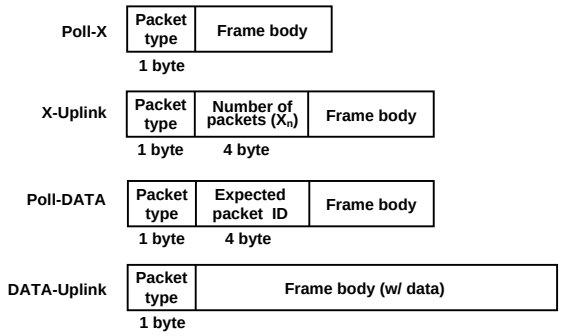

Figure 1: Packet Type

However, there are still some issues. Look at Figure 2, there is a possibility that AP Poll-X or a client poll X1 is not delivered successfully. Our method is retransmit Poll-X until the AP receives Xn. And we just set Xn = 0 if poll X does not deliver successfully.

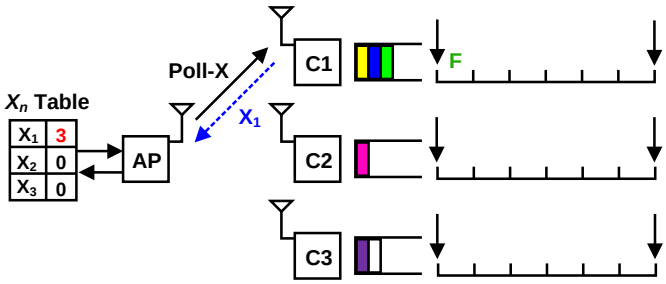

Figure 2: Issue 1

Look at Figure 3, how does a client know the DATA packet is delivered? Do we need an application-layer ACK for AP? We put expected packet ID in Poll-DATA packets so that the client knows its last packet is delivered when AP asks its next packet. Thus, we needn't an application-layer "ACK" for AP.

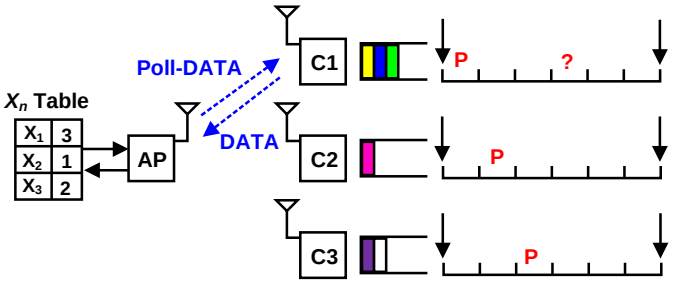

Figure 3: Issue 2

Look at Figure 4, what does "Xn" mean for non-real-time traffic? The answer is that the queue length

equals total number of packets generated by a client minus total delivery of data packets from a client. So the max-weight policy chooses a client with the largest channel reliability times current queue length.

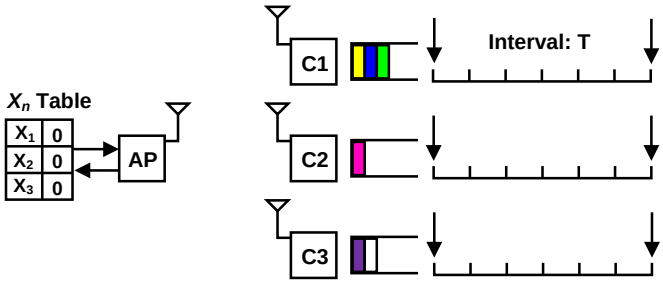

Figure 4: Issue 3

2 Implementation in NS-2

In our SWiFi project, the way we implement baseline policy is described as follows.

In the `command` function in swifi.cc

When the system captures a string called `server` indicating that it is a server(AP). If the value of `do_pull_num` is true, `poll_state_` will be set to `SWiFi_POLL_NUM`. Otherwise, when the value of `do_pull_num` is false, `poll_state_` will be set `SWiFi_POLL_DATA`.

When the system captures a string called `poll`, tcl will print an error: only server AP can poll if it's not a server. If it's a server, the server(AP) will ask each client one by one using function `scheduleRoundRobin` when the `poll_state_` is `SWiFi_POLL_NUM`. And we set the packet type to `SWiFi_PKT_POLL_NUM`, store current time and send the new packet. Otherwise, the server(AP) will use maxweight policy to schedule clients using function `scheduleMaxWeight` when the `poll_state_` is `SWiFi_POLL_DATA`. And we set the packet type to `SWiFi_PKT_POLL_DATA`, store current time and send the new packet. Besides, there is one special condition that no more client is left in one interval (`target_` is true). In this case, `poll_state_` will be set to `SWiFi_POLL_IDLE`.

When the system captures a string `boi` meaning at the beginning of each interval, the system will set `target_` to NULL. If `realtime_` is true indicating it is real-time traffic, we need clear number of data packets of all clients at the beginning of each interval.

In the `recv` function in swifi.cc

When a client receives a packet type `SWiFi_PKT_POLL_DATA` transmitted by a server, we save the old packet's send time, discard the old packet, create a new packet, set new send time and send this new packet to the server(AP). When a client receives a packet type `SWiFi_PKT_POLL_NUM` transmitted by a server, we set the data type to `SWiFi_PKT_NUM_UL` meaning that it is uplink packet with numbers of packets at client, set new send time and send this new packet to the server(AP). When a server receivers a packet type `SWiFi_PKT_NUM_UL` transmitted by a client, the current queue length is equal to the number of data packets generated minus the number of data packets received. For real-time traffic, the current queue length is equal to the number of data packets generated.

For Real-time traffic: At the beginning of each interval, we need clean queues of all clients cause we don't need store the queue length last interval. The current queue length just equals random number of packets at the beginning of each interval. If all packets miss its deadline, we just drop those packets.

For Non-real-time traffic: If all packets miss its deadline, we don't drop those packets. We need store the queue length last interval. The current queue length is equal to total number of data packets generated by clients minus total number of delivery data packets from clients.

We consider this system as a state machine. But NS-2 doesn't provide a state machine. We just define some enum to define these states.

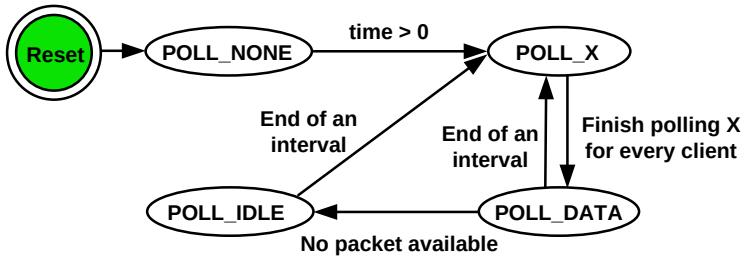

Figure 5: State Machine

3 Simulation Results

3.1 Symmetric System with 2 Clients

We assume T=10 intervals =100ms, channel reliability p1 = p2 = 1

For non-real-time traffic, according to Figure 6, `Nmax` ranges from 1 to 12. We can conclude that polling reduces channel capacity.

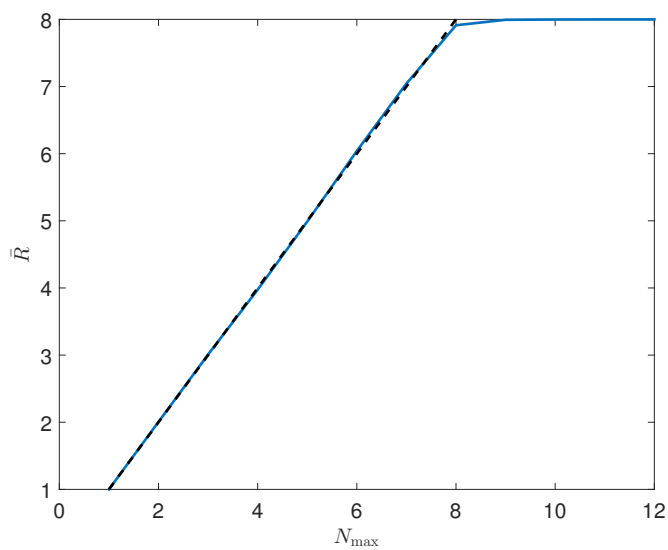

Figure 6: Non-real-time traffic and symmetric system with 2 clients

For real-time traffic, according to Figure 7, `Nmax` ranges from 1 to 20. We can conclude that packet deadline reduces the capacity further cause packets are dropped if they miss their deadline.

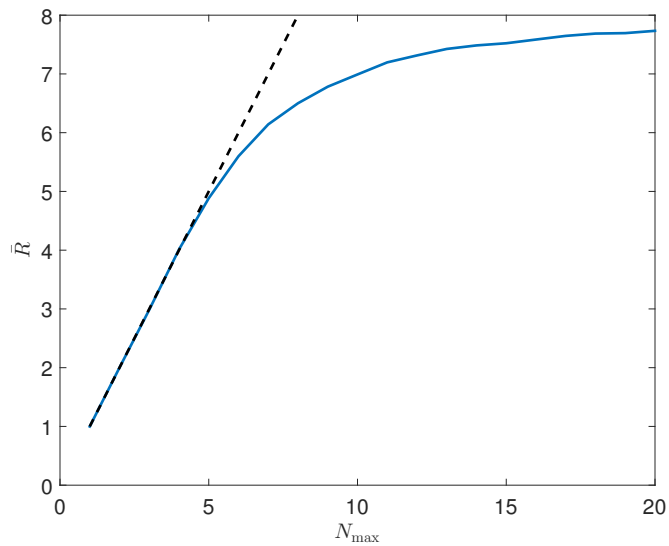

Figure 7: Real-time traffic and symmetric system with 2 clients

3.2 Asymmetric System with 2 Clients

We assume N = 2, channel reliability p1 = p2 = 0.57 (distance 1000m).

For non-real-time traffic, according to Figure 8, T ranges from 4 to 16. We can conclude that interval should be long enough to guarantee deliveries. If interval is too short, AP will not have enough time slots to poll data to clients.

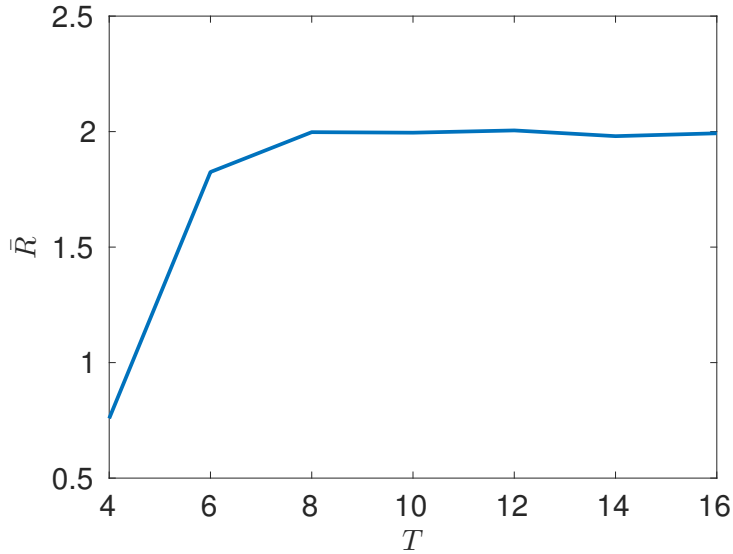

Figure 8: Non-real-time traffic and Asymmetric system with 2 clients

For real-time traffic, according to Figure 9, T ranges from 4 to 16. We can conclude that interval should be longer so that packets cannot expire and they won't miss their deadline.

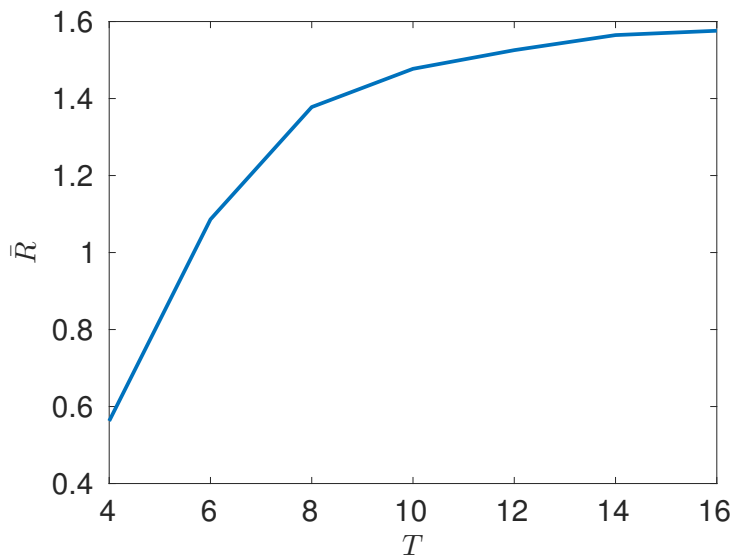

Figure 9: Non-real-time traffic and Asymmetric system with 2 clients

3.3 System with N Clients

We fix T = 10, channel reliability p1 = p2 = 0.57 (distance 1000m).

For non-real-time traffic, according to Figure 10, N ranges from 1 to 5. We can conclude that channel performance degrades severely with more clients. So the baseline policy is not suitable for a large number of clients cause it spends many time slots polling.

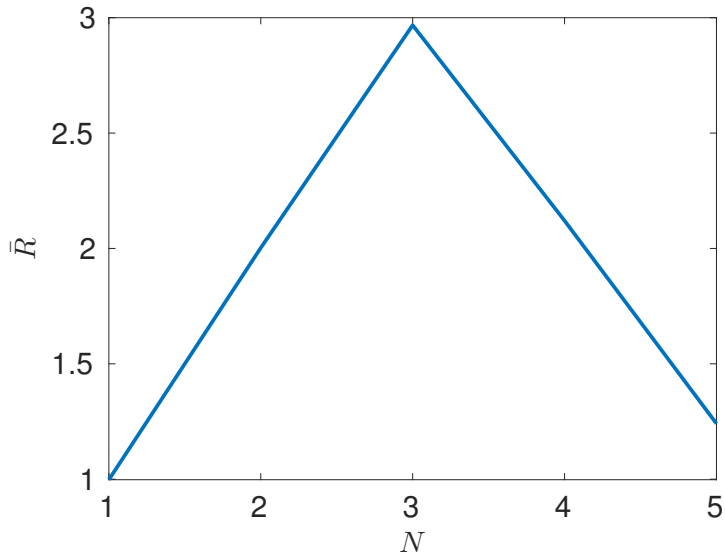

Figure 10: Non-real-time traffic with fixed T = 10

For real-time traffic, according to Figure 11, N ranges from 1 to 5. We can conclude that channel performance is even worse compared with non-real-time traffic and it also degrades with more clients. The reason is that packets will expire if they miss its deadline.

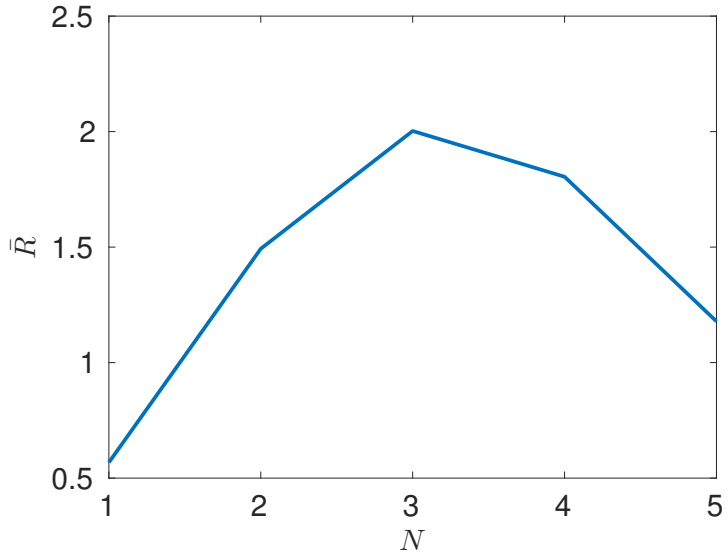

Figure 11: Real-time traffic with fixed T = 10

4 Conclusion

The system has a simple polling scheme and the server is work-conversing in scheduling clients based on maxweight policy. However, the channel utilization for data packets is low and it is not practical when the number of clients is large or the interval is small. The reason is that the system will spend many time slots on polling when it has a large number of clients N. Also, the system will not have enough time to poll if it has small interval T.

All in all, the baseline policy incus huge overhead. So we need a smarter policy.